(12)

INTEGRATING SYNTAX, SEMANTICS, AND DISCOURSE
DARPA NATURAL LANGUAGE UNDERSTANDING PROGRAM

R&D STATUS REPORT
Unisys/Defense Systems

Volume I -- REPORT

Deborah Dahl, John Dowding, Lynette Hirschman,
François Lang, Marcia Linebarger, Martha Palmer,
Rebecca Passonneau, Leslie Riley

May 14, 1987

INTEGRATING SYNTAX, SEMANTICS, AND DISCOURSE
DARPA NATURAL LANGUAGE UNDERSTANDING PROGRAM

R&D STATUS REPORT
Unisys/Defense Systems

Volume I -- REPORT

Deborah Dahl, John Dowding, Lynette Hirschman,
François Lang, Marcia Linebarger, Martha Palmer,
Rebecca Passonneau, Leslie Riley

May 14, 1987

INTEGRATING SYNTAX, SEMANTICS, AND DISCOURSE
DARPA NATURAL LANGUAGE UNDERSTANDING PROGRAM

R&D STATUS REPORT
Unisys/Defense Systems

Deborah Dahl, John Dowding, Lynette Hirschman,
François Lang, Marcia Linebarger, Martha Palmer,
Rebecca Passonneau, Leslie Riley

ARPA ORDER NUMBER: 5262
PROGRAM CODE NO. NR 049-602 dated 10 August 1984 (433)
CONTRACTOR: Unisys/Defense Systems
CONTRACT AMOUNT: $683,105
CONTRACT NO: N00014-85-C-0012
EFFECTIVE DATE OF CONTRACT: 4/29/85
EXPIRATION DATE OF CONTRACT: 4/29/87
PRINCIPAL INVESTIGATOR: Dr. Lynette Hirschman PHONE NO. (215) 648-7554

SHORT TITLE OF WORK: DARPA Natural Language Understanding Program

REPORTING PERIOD: 4/29/85 - 4/29/87

TABLE OF CONTENTS

TABLE OF APPENDICES

TABLE OF FIGURES

1. Overview

During the period 4/29/85 to 4/29/87, Unisys (formerly SDC) and New York University have pursued joint research on natural-language text understanding, as part of the DARPA Strategic Computing Program in support of the Fleet Command Center Battle Management Program (FCCBMP). During this time, the two groups have collaborated on Proteus-I, a natural-language text-understanding system that processes the remarks field of messages (*casualty reports* or CASREPs) about failures of starting air compressors (SACs). The goal of the research is to demonstrate a system capable of understanding paragraph-length messages in a restricted domain. Features of this joint Unisys/NYU system are listed below:

- Creation of an integrated system for the detailed understanding of text; the system includes modules for syntax, semantics, reference resolution, and temporal processing;
- Grammatical coverage providing the correct parse for 90% of CASREP sentences, including a full treatment of co-ordinate conjunction and an integrated treatment of sentential fragments characteristic of message traffic;
- Automatic production of a tabular summary of diagnostic information for sample CASREPs;

- Construction of a qualitative model of a starting air compressor;
- Use of the qualitative model to simulate normal and faulty behavior described by natural-language phrases.

Because this research has been conducted jointly at Unisys (Paoli) and NYU (New York), the two groups have focused on different modules within the system, merging each module into the system as it becomes stable. The Unisys effort has focused primarily on the semantic and pragmatic modules, while the NYU effort has been focused on development of a qualitative domain model (starting air compressors), use of this model in resolving noun phrase references, and qualitative reasoning about the fault-diagnosis process. The Unisys work, as specified in the original Statement of Work, has been conducted primarily in Prolog, to take advantage of the the excellent match between Prolog and the requirements for building a natural language processing system. This has resulted in a system with good performance and ready portability to a variety of hardware: the system runs on Vax, Sun, Explorer and Xerox Lisp Machines. The NYU work has been in Lisp and their completed system will be delivered in CommonLisp.

2. Objectives

The overall objective of the natural-language understanding work is to demonstrate capabilities for "understanding" information contained in free narrative. This understanding can be demonstrated in several ways: simulation of

events described in the narrative (as done by the current Proteus system); summarization of events described in the narrative (also done by the current Proteus system); or creation of database or knowledge base updates, to add information to an existing database or knowledge base. Such an understanding depends on the application of many sources of information, including syntactic, semantic, and pragmatic information, as well as detailed information about the specific domain in question. The work described in this report has focused on several critical research issues in building Proteus:

- Portability, supported by clear factoring of domain-independent and domain-specific information, and by a collection of tools to support creation of the various modules;
- Modularity, supporting incremental development of a large-scale system and permitting a division of labor between Unisys and NYU;
- Integration of multiple sources of information, to provide search focus during parsing and convergence on a correct interpretation of individual sentences (and ultimately of the entire discourse);
- Robustness, provided by a broad-coverage grammar, integration of multiple knowledge sources to detect inconsistent information, and feedback to the user to provide help in diagnosing missing or incorrect information;
- A development environment for the construction of a large-scale natural-language processing system, including tools for debugging, testing, updating, and tailoring the system to different types of development.

3. Status of Current Work

Proteus-I is a highly modular system consisting of separate syntactic, semantic, and pragmatic components. Each component draws on one or more sets of data, including a lexicon, a broad-coverage grammar of English, semantic verb decompositions, general mapping rules relating syntactic and semantic constituents, a domain model, and a model for diagnostic reasoning (see Figure 1 for the overall design of the system). Modularity, careful separation of declarative and procedural information, and separation of domain-specific and domain-independent information contribute to an overall system which is flexible, extensible and portable.

The system creates a set of event representations in predicate-argument form which represent the message content. From this set of information, it can generate several types of output. One is a summary of "significant events", shown in Figure 2 below. Another form of output is a relational database, which stores the set of events in the form of logical relations, along with their temporal relations. This form of output is discussed in Section 12. Proteus-I also incorporates a detailed structural and functional model of the equipment (the starting air compressor). This model is used in determining the structure and referents of noun phrases, in determining the relationship between events described in the CASREP (through a qualitative simulation of the equipment), and in displaying the equipment status. One of the displays produced from the equipment model is shown in Figure 3.

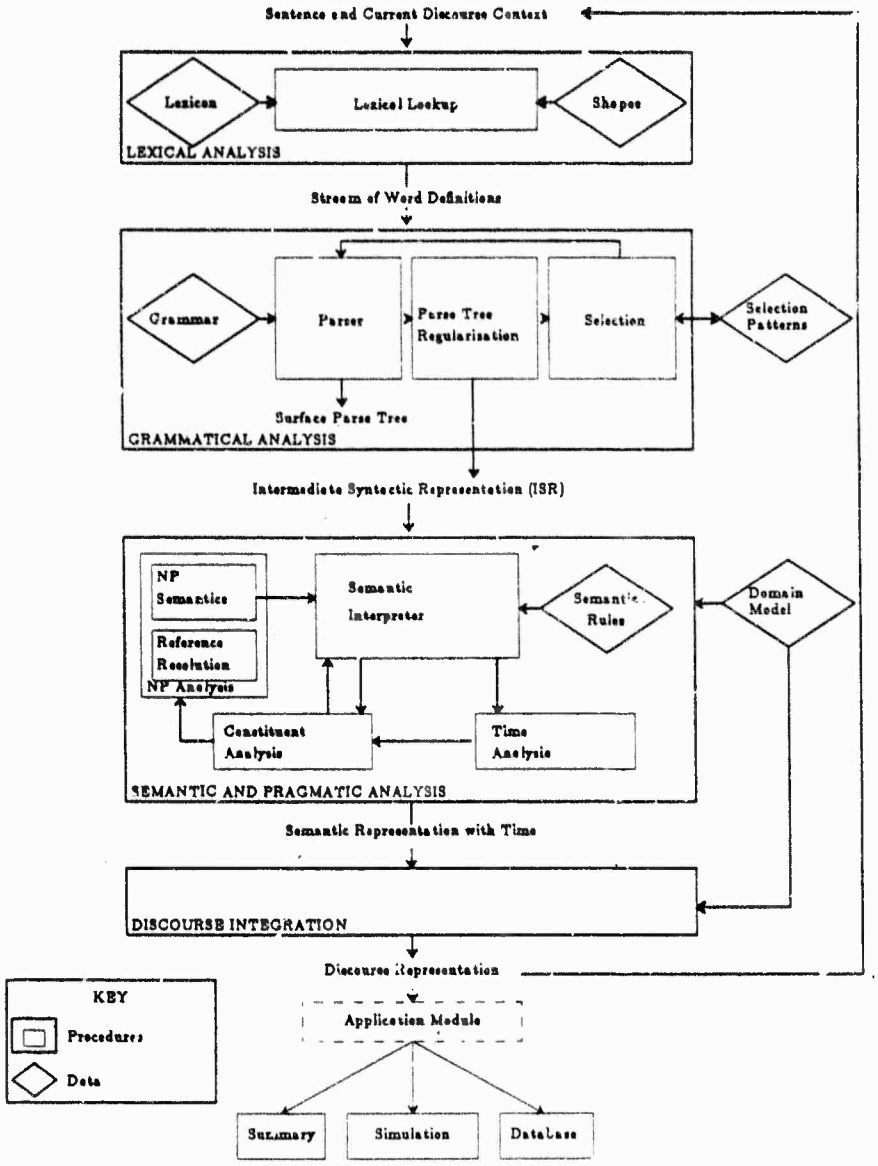

Figure 1.
Organization of Proteus-I

FAILURE OF ONE OF TWO SACS. UNIT HAD LOW OUTPUT AIR PRESSURE. RESULTED IN SLOW GAS TURBINE START. TROUBLESHOOTING REVEALED NORMAL SAC LUBE OIL PRESSURE AND TEMPERATURE. EROSION OF IMPELLOR BLADE TIP EVIDENT. CAUSE OF EROSION OF IMPELLOR BLADE UNDETERMINED. NEW SAC RECEIVED.

Status of Sac:
Part: sac State: inoperative

Finding:
Part: air pressure State: low

Finding:
Part: lube oil pressure State: normal

Finding:
Part: lube oil temperature State: normal

Damage:
Part: blade tip State: eroded

Finding:
Agent: ship's force State: has new sac

Figure 2.
Sample CASREP and Automatically Generated Summary

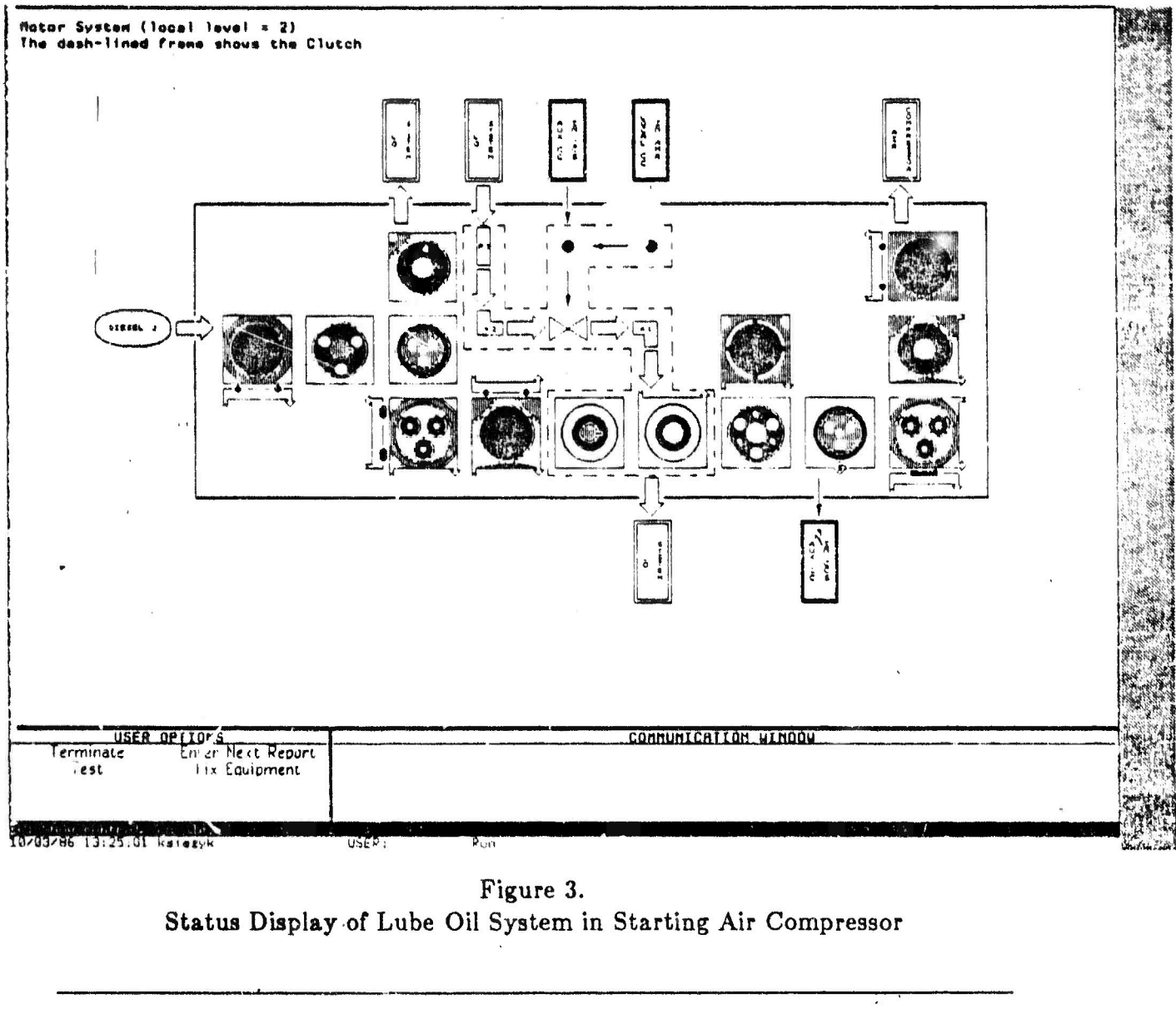

Figure 3.
Status Display of Lube Oil System in Starting Air Compressor

3.1. Status of the Unisys version of Proteus-I

As mentioned earlier, the Proteus-I system actually describes two systems, one in CommonLisp, being developed by NYU; and one in Prolog, being developed at Unisys. The remainder of this report will concentrate on the specific work being done at Unisys. The status of the Unisys version (called

PUNDIT) is as follows (refer to Figure 1 for a list of the components):

(1) The lexicon, developed by NYU, has been modified by Unisys to support handling of special "shapes" such as part numbers or dates; the lexicon contains all the words found in our sample corpus of 37 CASREPs (approximately 2000 lexical items). It is described in Section 4.

(2) The syntactic component (described below in Sections 5-6) consists of a broad-coverage grammar developed jointly by NYU and Unisys, a parser based on the Restriction Grammar work (developed under IR&D funding), including a general treatment of co-ordinate conjunction, a component for checking selectional constraints (developed by Unisys under a contract with NSF), and a regularization component developed by NYU and translated by Unisys into Prolog (under NSF funding, to support the work on selection).

(3) The semantic component consists of the semantic interpreter (translated into Lisp by NYU) that performs the analysis of several different types of predicating expressions, including clauses, nominalizations and noun predicates. The interpreter uses lexical semantic rules consisting of predicate decompositions as well as syntactic mapping rules and semantic class restrictions associated with the arguments of the predicates. The semantics component is described in Section 7, 8, and 9.

(4) A knowledge base (Section 3) contains the domain model. The knowledge base provides data for semantic and pragmatic analysis (and eventually,

for selection); at present, the Unisys system uses a limited domain model, but we are in the process of interfacing the complete NYU domain model (written in Flavors) directly to the Unisys Prolog system.

(5) The pragmatics or discourse component, developed at Unisys, includes components for reference resolution (Section 10) and temporal analysis (Section 11).

(6) Applications modules demonstrate "understanding" of the input text. Two application modules have been developed at Unisys. One generates a tabular summary of data in the CASREP, in order to highlight key information The second application module creates a database of critical information from processed CASREP messages. This database can be queried in natural language, using the PUNDIT system (see Section 12).

(7) The Unisys PUNDIT development environment (described in Section 13), which includes various special-purpose editors (for adding lexical, syntactic and semantic rules); and a testing procedure (described in Section 14), for testing additions to the overall PUNDIT system.

3.2. Summary of Key Unisys Contributions

The Unisys effort has been focused primarily on the development of a linguistically motivated, domain-independent framework providing integrated modules to extract information from a coherent discourse. The major areas of contribution from the Unisys research effort are:

- **A modular framework** for the processing of natural-language text, based on integration of information obtained from syntactic, semantic, and pragmatic analysis.
- A regular treatment of the kind of **fragmentary input** found in message traffic, including handling of omitted information, by means of a small set of extensions to the syntactic, semantic and pragmatic components for full text.
- **Well-defined interfaces between syntax, semantics and pragmatics,** based on two models of interaction: generate-and-test, and recursive call.
- **Portability,** based on careful separation of domain-specific and domain-independent modules.

The original statement of work outlines four major research areas: *coverage, portability, maintainability,* and *performance.* The major results from the current contract period are highlighted below.

COVERAGE

The PUNDIT system represents a significant advance in the state-of-the-art of automating message understanding. It is an ambitious attempt at comprehensive understanding of text, based on a complete syntactic analysis co-ordinated with semantic and pragmatic information from multiple sources. We have devoted considerable effort to investigating how to factor the processing into manageable modules and how to integrate these

modules for efficient computation. PUNDIT uses two distinct paradigms for interaction between the various components: 1) the "generate and test" paradigm, in which one component generates alternatives which are confirmed or rejected by another component, and 2) recursion, where components can be called recursively.

We have developed a broad-coverage grammar, based on the earlier work of Sager and colleagues at the NYU Linguistic String Project. Our grammar includes an efficient treatment of co-ordinate conjunction, and an integrated syntactic/semantic treatment of fragmentary input. The treatment of fragmentary input requires only a small set of extensions to the system for processing of full sentences, because it is based on the syntactic and semantic regularities found in the fragmentary input. On this basis, omitted information can be restored from context, and fragmentary sentences then processed as full sentences. The current system provides a correct parse for over 90% of the 132 sentences in our test corpus (see Appendix F for a detailed report on parsing coverage).

As the interface of syntax and semantics, we have designed a selection mechanism that *filters* semantically anomalous partial parses, using the "generate and test" approach: the parser suggests a possible parse for a sentence, the parse is regularized, and the selection component confirms or rejects this parse using a database of co-occurrence patterns allowed in the

specific domain (see Section 6). If the parse is rejected, an alternative parse is produced which is also passed on to selection. The regularized form of the final, presumably correct parse, is passed to the semantic component.

The control of the semantic component is closely interwoven with the control of the two pragmatic components, reference resolution and time analysis. The paradigm for interaction between noun phrase analysis and semantic analysis uses both the "generate and test" paradigm and recursion. At the top level, semantic analysis is called to perform a clause-level analysis, based on the clause predicate. At the point where a noun phrase is used to fill a semantic role associated with the verb, reference resolution is called to generate a possible referent for the noun phrase; the referent is tested against the semantic class restriction on the semantic role. If it is rejected, another referent is generated for testing, until a referent is finally confirmed. Semantic analysis uses this paradigm to control the interaction with reference resolution for pronouns and elided constituents as well as full noun phrases (see Section 9.4). The processing of nominalizations calls for recursion between clause semantics and noun phrase analysis: clause semantics calls noun phrase analysis to process a noun phrase containing a nominalized verb; then noun phrase analysis calls a version of clause semantics to process the predicate-argument relations that make up the nominalized expression.

The interaction between semantic analysis and time analysis also uses recursion. Semantic analysis calls time analysis after producing a decomposition for a phrase. Time analysis is responsible for processing time information from a variety of sources: tense, aspect, inherent verb semantics, and also temporal adverbial phrases. The processing of an adverbial phrase or clause may require a call to semantic analysis, which will, in turn, lead to another call to time analysis (see Section 11).

PORTABILITY

Our approach to portability has been to isolate the domain-specific information from the domain-independent information. The domain-specific information is regarded as *data* to the system, so that the basic architecture is portable from domain to domain. To support the process of adding domain-specific information, we have built tools to add lexical items and semantic rules. Domain-specific selection patterns (words co-occurring in syntactic relations, e.g., a subject-verb-object combination) are collected interactively during processing of the corpus. This allows for a "bootstrapping" into a new domain. The validity of this approach has been demonstrated by porting PUNDIT to a Navy ships domain under Independent R&D funding.

MAINTAINABILITY

Our major approach towards ensuring maintainability of the system has

been to provide a development environment which can be tailored to the needs of several classes of developers. This includes a set of trace packages and switches which allow various traces to be turned on or off (see Section 13). In addition, we have put in place a semi-automated test procedure, for ensuring functionality of successive releases of the system (see Section 14).

PERFORMANCE

Part of the motivation for our work on the integration of syntax, semantics and pragmatics has been to provide adequate performance. By using selectional information to filter parses, it is possible to focus much more quickly on the correct interpretation. Overall, the performance of the system, running under Quintus Prolog version 1.5 on a SUN 3, has been acceptable for development efforts -- processing time for a CASREP message ranges from 20 to 40 cpu seconds.

Research on Logic Grammars, conducted under the Unisys Independent R&D program, has also contributed to the efficiency of the current system. This work, described in Section 5.2, describes the concept of *Dynamic Translation*, which, in conjunction with rule pruning, has created an overall 20-fold speed-up in the syntactic analysis phase.

A related contract may also provide significant performance improvements. The natural-language application has become a major focus of current

Unisys work on parallel symbolic processing, supported under DARPA contract number F30602-86-C-0093 USAF/AFSC, Rome Air Development Center. Initial results under this contract lead us to believe that natural-language applications lend themselves well to exploitation of logical or-parallelism. This parallelism may well produce speed-up of at least an order of magnitude, using a very simple model of or-parallelism. The natural-language group will continue to interact closely with the group supporting the parallel symbolic architecture work, to pursue this promising path of research.

3.3. Structure of the Report

The remaining sections summarize work on the individual modules of the system (the lexicon, syntax, semantics, reference resolution, temporal analysis, application-specific output) and their interfaces (syntax-semantics interaction, semantics-pragmatics interaction), as well as the PUNDIT development environment and the testing procedure. The final section summarizes the focus of our future work during the two-year follow-on period.

For completeness in describing the PUNDIT system as a whole, the technical sections (Sections 4-14) also include work supported under Independent R&D funding and work supported under National Science Foundation grants DCR-8502205 and MCS-8202397. Where the work was *not* supported by DARPA, this is noted explicitly in the text.

4. Lexical Processing

The initial input to PUNDIT is a string of characters. This character stream is first fed to the *tokenizer*, which combines characters into *tokens.*[1] It is this token stream that is the input to the lexical processor, which produces a *definition stream* which is the input to the parser. The lexical processor associates with each token (or sequence of tokens in the case of multi-word expressions) a definition that is either found in the lexicon or can be inferred from the structure of the tokens. For instance, in certain domains the characters "(215) 648-1234" represent a telephone number (which can be classified as a noun), which must be recognized by the lexical processor.

The lexicon is the *data* associated with lexical processing. Our current lexicon contains several thousand entries related to the particular subdomain of equipment maintenance. The format of the lexicon is a modified version of the Linguistic String Project lexicon format [34], with words classified as to part of speech and subcategorized in limited ways (e.g., verbs are subcategorized for their complement types).

The lexical processor reduces morphological variants of a word to their root form. Information which is common to all of the morphological variants of a word is only stored on the root form, leading to a more compact lexicon.

[1] In programming language terminology this is called lexical analysis; however, to use that term in this setting would be misleading.

5. Syntax

This section on syntax covers both extensions to the Restriction Grammar framework (funded under IR&D) and grammar extensions to handle the CASREP corpus (done under funding from this contract). Syntactic processing in PUNDIT yields two syntactic descriptions of the sentence. One is an extremely detailed surface structure parse tree, and the other is a regularized operator-argument notation called the *Intermediate Syntactic Representation* (ISR), described in Section 6. The surface structure parse tree is the result of performing a detailed syntactic analysis of the sentence and is used to construct the ISR, which is a regularized input appropriate for semantics and selection. The grammar rules that are used by the parser are augmented BNF definitions, extended by rules for incremental computation of the ISR on the basis of its children. The part of the grammar that accounts for the surface syntax of the sentence is written in the *Restriction Grammar* (RG) formalism, while the part of the grammar which accounts for the Intermediate Syntactic Representation is written in the *Translation Rule Language* (TRL).

5.1. The Restriction Grammar

The Restriction Grammar formalism [14, 15] draws its grammatical approach from earlier work of Sager, Grishman, and Friedman in connection with the New York University Linguistic String Project [34, 33, 11]. The grammar is written in terms of context-free rules, augmented with context-sensitive restrictions stated as constraints on the partially constructed parse tree. The

parsing mecha ·.m is a standard top-down left-to-right parser. The parse tree is constructed incrementally after the successful application of each grammar rule, using a data structure that supports free tree traversal by the restrictions.

Restriction Grammar belongs to the rapidly expanding class of logic grammars, including Metamorphosis Grammar [3], Definite Clause Grammar [28], Extraposition Grammar [29], Definite Clause Translation Grammar [1], Modifier Structure Grammar [7], and Gapping Grammar [8]. The Restriction Grammar formalism shares with other logic grammars a set of context-free production rules interspersed with Prolog constraints. It extends the Definite Clause Grammar notion of "implicit" parameters to include not only the word stream, but also an automatically constructed parse tree. The Restriction Grammar differs from other logic grammars in that the constraints are restrictions on the well-formedness of the parse tree; the well-formedness of the tree is checked by routines that climb around the tree, inspecting its structure. In this way, Restriction Grammar avoids the standard DCG approach of relying on parameters in BNF definitions to pass around context-sensitive information.

Restrictions are written using a layered approach that makes the syntactic constructs independent of the particular implementation of tree structure. This approach has proved extremely useful in insulating the grammar from changes in the underlying execution mechanism. The lowest layer of operators consis' of primitive tree relation operators (such as **parent**, **child**, **left** and **right sibling**) and operators to extract the label of a node and the lexical item asso-

ciated with a node (also the lexical subclasses associated with a word). On top of this layer are a set of *restriction operators* that support various extended tree traversal operations (e.g., iterative ascent and descent), as well as operators to examine the word input stream for optimization purposes. The next layer of routines captures syntactic relations such as *head* of a construction, the *main verb*, or the *left/right adjunct* of a construction. Finally, restrictions (e.g., subject-verb agreement) are built out of the routines and the restriction operators.

Conjunction is handled by an extension to the basic Restriction Grammar framework that supports *meta-rules* ([16], also included as Appendix G). These meta-rules automatically rewrite "base" grammar rules into more complex rules to handle co-ordinate conjunction. The meta-rule component generates grammar rules specifying allowable conjoinings at limited types of nodes, to reduce redundancy. The resulting *Meta-Restriction Grammar* represents both the surface structure and a regularized structure (via pointers to elided elements) for efficient computation of selectional restrictions. This approach is sufficiently powerful to handle a number of complex phenomena, such as conjunction with comma (as distinguished from the appositive construction), paired conjunctions such as *both...and*, *either...or*, and scoping of left noun modifiers under conjunction. One of the great attractions of the meta-rule approach is that the grammar can be translated and *compiled*, resulting in an efficient treatment of conjunction. In PUNDIT, the current grammar consists of about 100 BNF

definitions before applying the conjunction meta-rule, and about 150 definitions after conjunction has been applied.

5.2. Parsing using the Dynamic Translator

As part of our Independent R&D work in the Natural Language Processing area, we were able to combine the concepts of translation and interpretation, to produce an approach we call *Dynamic Translation* ([9], also included as Appendix D). This work is described here because it is now incorporated into the overall system and contributes to its efficiency: dynamic translation, supporting rule pruning, has increased parsing speed by a factor of 20.

Traditionally, the parsing mechanisms for logic grammars have either been interpreters or translators. A (top-down) interpreter parses a string as a phrase of a given category by choosing a grammar rule of that category, dividing the phrase into sub-phrases, and parsing those sub-phrases into the designated categories. Thus, at runtime the interpreter requires not only the string that is to be parsed, but also the set of grammar rules. Alternatively, the process can be broken down into two phases, the translation phase and the runtime phase. The translator takes the complete set of grammar rules and produces a set of Prolog procedures which, when called at run time, will parse the phrase in exactly the same way that the original grammar would have. The translation phase converts the information explicit in the grammar rules into information implicit in the Prolog procedures. The translation phase requires the set of grammar rules but does not have the

input string available to it, while the runtime phase after translation has access to the input string, but does not have explicit access to the original grammar rules. Because of this loss of explicit information when moving from interpretation to translation, the translated Prolog code will be more efficient than the interpreter, but the interpreter will be more flexible.

In contrast to these traditional approaches, the Restriction Grammar uses a single mechanism, the Dynamic Translator, that takes advantage of the strengths of translation and interpretation without the corresponding disadvantages. The Dynamic Translator has available to it both the input string and the grammar rules (like an interpreter), but also makes use of both a translation and a run time phase (like a translator). In the Dynamic Translator, the translated code runs in co-operation with an interpreter to parse a sentence. Although one might thus expect that the speed of the Dynamic Translator to be intermediate between an interpreter and a translator, the Dynamic Translator is substantially faster than either. This added efficiency is gained by the use of the Dynamic Rule Pruning mechanism, an inherently interpretive device that dynamically prunes the search space.

The Dynamic Rule Pruning mechanism uses information available at runtime to reduce the number of options that must be considered for certain grammar rules. This information includes, for example, both the input word stream and the partially constructed parse tree. Reducing the number of options eliminates extraneous paths from the search space and

greatly increases the efficiency of the parsing process (measured as a factor of 20 in parsing the sentences from part of our CASREP corpus).

5.3. Grammar Enhancements for the CASREPs

Both the BNF and the restriction components of the grammar have been extended considerably in connection with work on the CASREPs.

One area of development has been the treatment of sentence fragments. Approximately half of the sentences in the CASREPs are written in the "telegraphic" style characteristic of message traffic. Nevertheless, these fragments follow quite regular patterns, which characterize deliberately telegraphic speech in other subdomains as well.

The fragments found in the CASREPs fall into one or another of five basic types:

zerocopula:

a subject followed by a predicate, differing from a full clause only by the absence of the verb *be*, as in *Impellor blade tip erosion evident.*

tvo:

a tensed sentence (tensed verb + object) missing its subject, as in *Believe the coupling from diesel to sac lube oil pump to be sheared.*

nstg_frag:

an isolated noun phrase (noun-string fragment), as in *Loss of oil pump pressure.*

objbe_frag:

an object-of-be fragment is an isolated complement of auxiliary verb *be*, *Believed due to worn bushings*, where the full sentence counterpart is *Failure is believed (to be) due to worn bushings.*

predicate:

is an isolated complement of the main verb *be*, as in *Unable to consistently start nr 1b gas turbine.*

Note that sentences with missing determiners (as in *sac has failed*) are *not* counted as fragments, since omitted determiners are characteristic of practically every sentence in our corpuses.

For the most part, these ellipses are filled in at the level of the Intermediate Syntactic Representation rather than in the surface parse tree. Therefore a full discussion of fragments is deferred to the following section on the ISR.

It should be observed that other types of fragments, which we have not yet encountered, may occur in this domain. Missing objects, as in *Factory should replace immediately*, represent a fragment type found in other "telegraphic" domains, along with other degenerate structures, such as preposition-less prepositional phrases (*Tumor found left lower lung*). However, such ellipses have not been encountered in the current corpus.

The grammar has also been extended to cover a wider range of object types, including a variety of embedded infinitivals (discussed further in the following section), embedded clauses, and "small clauses" such as **sobjbe** (subject

+ object of be), as in *Ship's force found sac inoperative.* In addition, a rich variety of sentence adjuncts occur in the CASREPs, including a range of clausal and sub-clausal strings introduced by subordinating conjunctions and present participles. These extensions to the BNF rules have led to corresponding expansion of the restriction component, in order to prevent spurious ambiguities from arising because of enriched syntactic coverage.

6. Interaction of Syntax and Semantics

The interactions between the syntactic and semantic modules are, of course, bidirectional. The semantic interpretation of a sentence is constrained by its syntactic structure; and, conversely, the timely use of semantic information can dramatically reduce the syntactic search space. The use of syntactic information by the semantic module is carried out primarily by the mapping rules which assign thematic roles to arguments of the verb on the basis of their syntactic role. These rules are discussed in Section 7; in [22], included as Appendix B; and in Appendices A and K. The exploitation of semantic information to limit the syntactic search space is largely the function of the selectional patterns which are created through interaction with the user; this selectional component is described in greater detail in Section 6.3 below.

The work on the ISR, specifically the translation of the Lisp version of the ISR developed originally at NYU, was funded under the NSF contract DCR-8502205, because it was in direct support of the implementation of the selectional mechanism proposed under that contract. The extension of the ISR to

the CASREP domain was done under DARPA funding (Section 6.2); the implementation of the selectional component (Section 6.3) was completed under NSF funding.

These two areas of interaction are not independent of one another. The selectional patterns provide a map for the development of the deeper semantics, since they provide evidence about even quite fine-grained semantic dependencies. Conversely, the semantic component can be applied to generate selectional patterns on the basis of a domain model, if one is available, and to GENERALIZE lexical selectional patterns. Consider, for example, a lexical pattern such as the information that *the flame* cannot be the subject of the intransitive verb *melt* (as in *The flame melted*), although it may occur as the subject of a transitive verb (as in *The flame melted the plastic cup*). This pattern occurs only with certain verbs; in contrast to *melt*, for example, *eat* imposes the same selectional constraints on its subject in both transitive and intransitive uses. (The unacceptability of *The pan ate* predicts the unacceptability of *The pan ate lunch.*) These differences between *eat* and *melt* follow from the different ways in which the two verbs assign thematic roles to their subjects: *melt* assigns the role of **theme** to the subject of the intransitive, **agent** to the subject of the transitive; whereas *eat* assigns the role of **agent** to its subject in both its transitive and intransitive uses.

Thus from the information that *The pan eats* fails selection, the semantic component can generate *ALL* those patterns in which *the pan* plays the same

thematic role (**agent**); the patterns generated will include both transitive and intransitive occurrences of *eat* with *the pan* as subject. In contrast, the failure of *The flame melts* will not be generalized to exclude transitive sentences like *The flame melts the cup*, since *the flame* is assigned a different thematic role when it occurs as the subject of *melt* in its transitive use. From the information that *The flame melts* fails selection, the semantic component will be able to predict that any structure in which *the flame* is assigned the role of **theme** will be similarly unacceptable. *Something melts the flame*, for example, can now be excluded. In addition to extending lexical patterns on the basis of thematic role assignment, the semantic component will also be able to generalize selectional patterns across semantic classes of verbs and nouns; a selectional pattern generated for *drop* can be extended to *fall*, *decrease*, and so forth.

For all of these interactions between syntax and semantics, the vehicle of communication between the two modules is the INTERMEDIATE SYNTACTIC REPRESENTATION (ISR), which regularizes, and in some cases expands the surface structure parse tree into a representation of just those aspects of syntactic structure that are relevant to semantics and selection. The ISR serves as input to both the selectional component and the semantics. This level of representation is described below.

6.1. Intermediate Syntactic Representation

The ISR is computed for a sentence by annotating each node in the parse tree with an expression in the Translation Rule Language (TRL). These expressions dictate how the ISR of the children of a node can be combined to form the ISR of the parent. The ISRs of the nodes at the frontier of the tree come from the dictionary.

The Translation Rule Language is a simple programming language for manipulating syntactic descriptions. In a grammar rule, the TRL is given on the right hand side of a "->" symbol, with the Restriction Grammar notation on the left hand side. (These two notations can be freely intermixed, provided that the parent node is assigned exactly one TRL expression.) A simple example is the grammar rule for **sentence**:

```
sentence ::= center, ([.] ; [?]) -> center.
```

This rule says that a sentence node has two children nodes, one of which is a center node and the other is either a period or a question mark. (In the RG formalism, commas represent conjunction, semi-colons represent disjunction, "*x" represent lexical categories (of type **x**), and brackets surround literal expressions.) The TRL expression for **sentence** is just **center**, which indicates that the ISR of **sentence** is simply taken from the ISR of its **center**.

The ISR rule for **assertion** creates a VSO list:

```
assertion::= sa, subject, sa, ltvr, {wagree}, sa, object, sa ->
```

```
[ ltvr, subject, object, !sa*, -ltvr].
```

The head of the verb-adjunct complex (ltvr) is fronted, and the adjuncts are ordered at the end of the list following the sentence adjuncts (sa); the latter are spliced together at the end of the ISR list regardless of their position in the surface clause. More complex ISR representations will be detailed in the discussion below.

The ISR is computed only when it is needed by either semantics or selection in order to minimize the amount of work done. The alternative would be to compute the ISR of every node when it is finished being built (eagerly). We have decided against this approach because many nodes built in the course of parsing are discarded before they are ever considered by selection or semantics. By not computing the ISR of nodes that may never be needed (e.g., by lazy evaluation of the ISR) we save considerable overhead during parsing. Currently, the ISR is only computed at the completion of the noun phrase, assertion, fragment, question, and sentence nodes.

6.2. ISR Enhancements for the CASREPs

The purpose of the Intermediate Syntactic Representation is to provide a regularized structure as input to those components of the system which enforce selectional restrictions and develop a final representation of the information content of the sentence. This regularizing involves both the *elimination* of structural information not relevant to these modules, and the *addition* of other

information which is less explicitly represented in the syntactic parse tree. These two processes are illustrated in connection with the ISR treatment of auxiliaries, fragments, passives, and infinitival structures.

6.2.1. Auxiliaries

In the surface syntax of a sentence, the main verb may occur in a variety of forms and may be preceded by any number of auxiliary elements, as in *He has been repairing the pump*, with main verb *repair*. In the ISR of this sentence, however, tense and aspectual information is extracted from these auxiliaries and precedes the main verb. The verb is listed in its uninflected form, followed by its subject and object. Each noun phrase appears as a list consisting of the determiner (**tpos**), followed by a list containing the head noun (**nvar**) and its modifiers, if any. The head noun is itself a complex structure, consisting of its uninflected form, followed by number and a unique identifier, indicated below by a capital letter[2].

```
[present,perf,prog,repair,
    subject([pro([he,singular,X])]),
    object([tpos(the),[nvar([pump,singular,Y])]])].
```

Thus the tense and aspectual markers are extracted and precede the verb, subject, and object. When *be* is used as an auxiliary, it does not appear in the ISR; *The pump is failing*, for example, receives the following ISR:

[2] We will adopt the convention of explicitly labelling the subjects and objects in the ISR for the sake of clarity,

```
[present,prog,fail,
    subject([tpos(the),[nvar([pump,singular,X])]])]
```

In contrast, *be* does appear in the ISR of *The pump is full*, since it functions as the main verb in this case:

```
[present,be,
    subject([tpos(the),[nvar([pump,singular,X])]]),
    object(adj([full]))]
```

Extracting the main verb and factoring out its operators facilitates selection and semantic interpretation, and provides more appropriate input to the module which computes temporal relationships.

6.2.2. Fragments

The ISR treatment of fragments provides an example of the augmentation of syntactic information in the ISR. As noted above, the surface parse tree for a sentence fragment does not, in most cases, resemble that of a full assertion. It is at the level of the ISR that the fact that a subject or verb has been elided is made explicit. The ISR treatment of each of the five fragment types is considered briefly below.

Tvo: A subjectless tensed clause such as *Operates normally* is mapped onto an ISR of the following form, in which the missing subject is filled in by the

---

and of italicizing the ISR representation of lexical items.

dummy element **elided**:

```
[present,operate,
    subject(elided),
    adv(normal)]
```

At the level of the ISR, then, the fragment differs from a full assertion such as *Sac operates normally* only by virtue of this element **elided**, which is added in the ISR in the position of the missing subject. The element **elided** is assigned a referent by reference resolution exactly as if it were a pronoun; that is, as if the surface structure had been *He/she/it operates normally.* (Reference resolution is discussed more fully in Section 10 below.)

Zerocopula: As noted above, the surface parse tree provides a null verb for this fragment type; this null verb is replaced by *be* in the ISR if it is a main verb. Thus *Disk bad* receives the ISR

```
[untensed,be,
    subject([tpos([]),[nvar([disk,singular,X])]]),
    adj([bad])].
```

This ISR is nearly indistinguishable from that assigned to the corresponding full assertion *Disk is bad.* The only difference is that the *be* in the fragment is marked as **untensed**, but in the assertion it has the tense marker **present**. If the null verb represents auxiliary *be*, as in *Sac failing*, then it does not appear in

the ISR:

```
[untensed,prog,fail,
    subject([tpos([]),[nvar([sac,singular,X])]])].
```

Thus the null verb inserted in the syntax is treated in the ISR in the same way as overt occurrences of *be*.

Nstg_frag: The syntactic parse tree for an **nstg_frag** contains no empty elements; it is a regular noun phrase, labelled as an **nstg_frag**. The ISR bears the full burden of transforming it into a VSO sequence. This is done by building an ISR in which the noun phrase is the subject of an element **empty_verb**; in the semantic component, the subject of **empty_verb** is treated as the sole argument of a predicate **existential(X)**. As a result, the **nstg_frag** *Failure of sac* and a synonymous **assertion** such as *Failure of sac occurred* are mapped onto identical *semantic* representations by virtue of the semantics of this element **empty_verb**: Both are eventually mapped onto a semantic representation of the form

```
becomeP(inoperativeP(patient([sac7])))
```

even though the two sentences differ in their respective parse trees and ISRs. (For a discussion of the semantic processing of these structures, see Section 9 and [22, 23], included as Appendices B and K.

Objbe_frag and **Predicate**: These two fragment types are transformed into VSO sequences by insertion of the same elements used for **zerocopula** and **tvo**. The surface parse tree of these fragment types contains no empty elements; the untensed verb *be* (main or auxiliary) is inserted into the ISR, along with the dummy subject **elided**; as described above, reference resolution supplies the referent for **elided**. Thus the simple adjective *Inoperative* will receive an ISR with a VSO structure containing main verb *be*:

```
[untensed,be,
    subject(elided),
    adj([inoperative])].
```

On the other hand, *be* does not appear in the ISR of *repairing sac* (predicate), since it functions as an auxiliary in the corresponding assertion.

```
[untensed,prog,repair,
    subject(elided),
    object([tpos([]),[nvar([sac,singular,X])]])]
```

6.2.3. Passive

In order to provide selection with a canonical VSO input, passive sentences such as *The pump is being repaired* are regularized to a form in which the surface subject becomes the underlying object. The subject position is marked with the dummy element **passive** in the ISR, and semantics rules for passive struc-

tures look in the *by*-phrase for the agent (or whatever thematic role would be assigned to the subject of *repair* in the active):

```
[present,prog,repair,
    subject(passive),
    object([tpos(the),[nvar([pump,singular,X])]])]
```

This regularization allows the selection mechanism to check verb-object patterns with no special machinery for passive sentences.[3]

6.2.4. Null Subjects Of Infinitives

As a final example of the expansion of structure in the ISR, consider the treatment of sentences such as the following:

(1) They told the dentist to examine the student.

(2) They believed the dentist to have examined the student.

The linguistic literature draws a distinction between these two cases, although the formalisms by which this distinction has been captured vary greatly. In (1), *the dentist* is the **patient** argument of *tell* (the one ordered to do something), which also assigns the role of **theme** to the proposition *the dentist examine*[*s*] *the student*; it is also the **agent** of *examine* (the one examining) in the embedded infinitival. In (2), by contrast, the sole object argument of *believe* is the pro-

[3]Note that passive is not "undone" fully, since a noun phrase in a *by*-phrase is not inserted into subject position

position *the dentist examine*[*s*] *John. The dentist* plays only the role of **agent** of *examine*; it is not a **patient** of *believe.*

In PUNDIT, this distinction is only implicit in the surface parse. The two complements are assigned different BNF labels: The complement of *believe* in (2) is labelled **ntovo**, while the complement of *tell* in (1) is labelled **objtovo**. However, the actual difference in argument structure is made explicit at the level of the ISR, which assigns to **objtovo** an ISR in which a copy of the subject of the infinitive is created, so that this NP functions both as object of the matrix verb and subject of the embedded infinitive. The ISR for (3) is thus (4), in which the pronoun *him* functions both as the object of the verb (the "tellee") and as subject of the infinitival, which is itself an object of the verb. The two positions are co-indexed, as indicated by the identical referential index **Y**:

(3) They told him to examine the sac.

(4) [past,*tell*,

subject([pro([*they*,plural,X])]),

object([pro([*him*,singular,Y])]),

compl([untensed,*examine*,

subject([pro([*him*,singular,Y])]),

object([tpos(*the*),[nvar([*sac*,singular,W])]]]])]

In contrast, an ntovo such as (5) receives ISR (6), in which the matrix verb has

in the ISR. Thus subject-verb selectional patterns are treated somewhat differently than in active sentences.

only one object argument, the infinitive itself:

(5) They expected him to examine the sac.

(6) [past,*expect*,

subject([pro([*they*,plural,X])]),

object([untensed,*examine*,

subject([pro([*him*,singular,Y])]),

object([tpos(*the*),[nvar([*sac*,singular,Z])]]]])]

Thus the ISR in some cases enriches rather than strips down the surface parse, providing a more adequate input to selection and to semantic processing.

6.3. Selectional Restrictions in Parsing

One function of the ISR is to serve as input to the selectional component, which is a mechanism designed to block semantically ill-formed parses. We present in this section one of our approaches to ruling out such semantically anomalous parses.

A general problem encountered in parsing with large, broad-coverage grammars is that such grammars often produce a great number of parses. Our restriction grammar, for example, produces over 25 parses for five of the sentences in one of our corpuses. A majority of these parses, however, are incorrect because they violate some domain-specific or commonsense semantic constraint.

For example, two of the parses for the sentence *High lube oil temperature believed contributor to unit failure* could be paraphrased as:

(1) The high lube oil temperature believed the contributor to the unit failure.

(2) The high lube oil temperature was believed to be a contributor to the unit failure.

but our knowledge of the domain (and common sense) tells us that the first parse is wrong, since temperatures cannot hold beliefs.

It is only because of this semantic information that we know that parse (2) is correct, and that parse (1) is not, since we cannot rule out parse (1) on syntactic grounds alone. In fact, our grammar generates the incorrect parse before the correct one, since it produces assertion parses before fragment parses. If a parser has access to domain knowledge of this kind, however, many incorrect parses such as (1) will never be generated. The selection component has been designed to collect this sort of domain information and store it in the form of allowable and anomalous lexical co-occurrence patterns.

The essential feature of our parser which enables the collecting of such syntactic co-occurrence information is the ISR produced by the syntax processor, which, as we have seen, is the result of regularizing the surface syntactic structure into a canonical form of operators and arguments. Since the ISR regularizes syntactic patterns into a canonical form consisting of predicate-argument

and head-modifier structures, there are only a fairly limited number of patterns which can appear in an ISR. Under contract DCR-8502205 supported by the NSF, we adapted the ISR developed by NYU to Prolog for PUNDIT, and have written a program to analyze the ISR and examine (and store) syntactic patterns as they are generated. By using this program while parsing many sentences in a corpus, we have collected a large number of such syntactic co-occurrence patterns. In addition, (although this is a bit of an oversimplification) by noting which parses are correct, we have further divided the patterns collected into those representing relationships which can hold among domain entities, and those which cannot. Finally, giving the parser access to these patterns enables it to automatically fail parses such as (1) above. This selectional mechanism has reduced the average number of parses generated in a 31-sentence corpus from 4.7 parses per sentence to 1.5, and decreased the average amount of time spent parsing by over one-third. In parsing these 31 sentences, 284 selectional patterns were collected, of which 171 were good, and 113 bad.

6.3.1. Some Typical Selectional Patterns

A typical pattern generated by a predicate-argument structure is the SVO pattern, which consists of the subject of a sentence, the sentence's main verb, and its object. In most cases, the subject and object will be represented in the SVO pattern by a specific lexical item; clausal structures functioning as subject or object, however, are represented simply by the special atom CLAUSE. For

example, the sentence *The field engineer repaired the turbine* would generate the SVO pattern [*field-engineer repair turbine*], while the sentence *The field engineer believes that the turbine has been repaired* would generate the pattern [*field-engineer believe* CLAUSE]. A typical head-modifier pattern is the ADJ pattern, which consists of the head noun of an NP and an adjective modifying that noun. The NP *metallic particles* generates the ADJ pattern [*metallic particles*]. A few other common patterns are the CONJ pattern, consisting of a conjunction surrounded by its two conjuncts (e.g., *Troubleshooting revealed normal pressure and temperature* yields the CONJ pattern [*pressure and temperature*], as well as the SVO patterns [*troubleshooting reveal pressure*] and [*troubleshooting reveal temperature*]); and the ADV pattern, in which are found an adverb and the verb it modifies (e.g., from the sentence *Compressor disengaged immediately* we get the ADV pattern [*disengage immediate*]).

Let us take as an example the sentence discussed above: *High lube oil temperature believed contributor to unit failure.* In the ISR of the correct parse for this sentence (in which the verb *believe* is a passive), the atom SOMEBODY/THING is used as the placeholder for the subject, and the atom CLAUSE represents the small clause *oil* [*be*] *contributor*, which is the object. The SVO pattern corresponding to this sentence is therefore [SOMEBODY/THING *believe* CLAUSE]. Recall that this parse is the second one found. In the incorrect (and somewhat amusing) reading which is generated first, the verb *believe* is active, *temperature* is the subject, and *contributor* the direct object,

creating the pattern [*temperature believe contributor*]. This is the pattern that is rejected on the basis of temperatures not being able to hold beliefs.

Ruling out semantically anomalous patterns could also be accomplished by using the arguments of the semantic decomposition produced by the semantic analysis component (discussed in Section 7). For example, in the semantic analysis component, the verb *believe* is represented as having two arguments, an **experiencer** and a **theme**. The **experiencer** is given a semantic class restriction of being **animate**, and the **theme** is given a semantic class restriction of being a **proposition**. In our example, the semantic analysis would eventually reject *temperature* as a potential **experiencer** for the *believe* representation, since *temperature* would fail the **animate** semantic class restriction. By way of contrast, in the second parse, SOMEBODY/THING is unmarked for animateness, and so would be compatible with an **animate** semantic class restriction. In this way, the deep semantic analysis will also be able to reject the incorrect parse.

Thus the incorrect parse of this sentence could be rejected by either the selection mechanism or the semantic component. Both approaches to constraining the syntactic search space are under development; in particular, we are currently studying the design of a more flexible control structure for the interaction of syntax and semantics which would allow the semantic analysis procedure to reject anomalous parses *during* rather than *after* the assertion parse. (This mechanism is discussed in Appendix N.) However, efficiency considerations would tend to argue for the employment of selectional patterns rather than for

transfer of control to semantics at this early point in the parse, especially since the in-depth semantic analysis performed by the semantic interpreter may well turn out to be premature in the event that the parse under construction fails on purely syntactic grounds. The goal of current work is therefore to bring the selection mechanism and the semantic component into closer co-operation: to use the semantic module to generalize selectional patterns acquired through interaction with the user, and to use the selection mechanism to filter bad parses on the basis of constraints embedded in the semantic interpreter.[4]

6.3.2. The User Interface

The selectional pattern analyzer is invoked by two restrictions which are called after the BNF grammar has assembled a complete NP (and constructed the ISR for that NP), and after it has assembled a complete sentence (and constructed its ISR). The program operates by presenting to the user a syntactic pattern found in the ISR, and querying him/her about the acceptability of that pattern. When presented with a pattern, the user can respond to the query in one of two ways (explained in detail below), depending on the semantic compatibility of the head and modifiers (e.g., for an ADJ pattern) or of the predicate and arguments (e.g., in the case of an SVO pattern) contained in the pattern.

One possible response to the program's query is to reject the pattern. This is the appropriate action if the pattern presented describes a relationship that

[4]For example, the semantic class restrictions on *believe* could be employed to produce a general semantic pattern such as [**animate** *believe* **proposition**], which would also rule out the incorrect parse paraphrased in (1).

cannot hold among domain entities. Rejecting a pattern classifies it as bad, and signals an incorrect parse. For example, after the parser has constructed the ISR corresponding to the first parse of the sentence discussed earlier (*High lube oil temperature believed contributor to unit failure*) and passed it tc the selectional pattern checker, the program queries the user about the SVO pattern [*temperature believe contributor*].[5] The query to the user consists of two parts: the type of pattern (in this example, SVO), and the lexical items (or, in certain cases, the special atoms, such as CLAUSE) forming the instance in question of the pattern [*temperature believe contributor*]. Thus, in our example, the message to the user reads

```
SVO : temperature believe contributor
```

In presenting this pattern to the user, the program is asking whether the noun *temperature* can be the subject of the verb *believe* with *contributor* as the direct object. The user can then respond by typing either *yes* or *no*, depending on whether or not the pattern is a good one. In the current example, the user would fail the pattern (by typing *no*), since in our domain model (and in the real world) one cannot speak of temperatures believing things. This response causes the selection restriction to fail, and as a result, the parse under construction is immediately failed, and the parser backtracks.

[5] In this simplified explanation, we present only the SVO patterns. In actual parsing of this sentence, however, additional patterns would be generated from the NP level.

The other possible response to the program's query is to accept the pattern. This is the appropriate response if the pattern presented describes a relationship that can be said to hold among domain entities. Accepting a pattern classifies it as good, and allows the analysis of the ISR to continue. In our running example, after the first parse for the sentence fails, the parser then tries to generate the second parse for the sentence, and presents the pattern

```
SVO : SOMEBODY/THING believe CLAUSE
```

asking whether it is reasonable to speak of some unspecified subject believing a proposition expressed by a clause. The user would accept this pattern, since in the domain propositions expressed by clauses can be objects of belief. Since the pattern was judged good, the analysis of the ISR (and the parsing of the English sentence) is then allowed to continue.

As the user classifies patterns into "good patterns" and "bad patterns", they are stored in a pattern database which is consulted before any query to the user is made, so that once a pattern has been classified as good or bad, the user is not asked to classify it again. If a pattern previously classified as bad by the user (and therefore in the DB) is encountered in the course of analyzing the ISR, the program consults the database, recognizes that the pattern is bad, and automatically fails the parse being assembled. Similarly, if a pattern previously recorded as good is encountered, the program will recognize that the pattern is good simply by consulting the database (and not querying the user), and

allow the parsing to proceed.

Additional statistical information about the effect of the selectional mechanism and a more detailed discussion of the methodology can be found in Appendix O.

7. Semantic Analysis - Basic Approach

The semantic analysis approach used by PUNDIT has two separate, but inter-related components: 1) the algorithm that controls the execution of the rule-driven semantic analysis and 2) the theory of lexical semantics captured by the rules themselves. The basic approach described below was originally designed for the analysis of main clauses where the PREDICATING EXPRESSION was the verb. It was an independent system that assumed the existence of separate components to parse sentences and to resolve referents of noun phrases, and performed a very rudimentary time analysis. The implementation of PUNDIT has caused the rule-driven semantic analysis approach to be fully integrated with a syntactic parser, a reference resolution component, and a sophisticated time analysis component. In addition, the analysis algorithm has been extended to cover predicating expressions in a full range of syntactic environments, including noun phrases and modifiers as well as verbs. The CASREP domain has also made demands on the theory of lexical semantics, which has been regularized to follow more traditional linguistic classifications for the semantic roles, and has been extended to cover a much broader range of verb subcategorizations. This section will outline the basic approach to

semantic analysis that PUNDIT uses. The next section, Section 8, describes the lexical semantics for the CASREPs domain, and Section 9 explains the extensions to the algorithm occasioned by the increased interaction with syntax and pragmatics.

Semantic analysis in PUNDIT is based on Inference Driven Semantic Analysis [22] which decomposes verbs into component meanings and fills their semantic roles with syntactic constituents. The result of the semantic analysis is a set of PARTIALLY instantiated semantic predicates which is similar to a frame representation. To produce this representation, the semantic components share access to the DOMAIN MODEL that contains generic descriptions of the domain elements corresponding to the lexical items. The model includes a detailed representation of the types of assemblies that these elements can occur in. The semantic components are designed to work independently of the particular model by relying on an well-defined interface.

In order to produce the correct semantic representation, the predicating expression is first decomposed into a semantic predicate representation appropriate for the domain. The arguments to the predicates constitute the SEMANTIC ROLES of the predicating expression, which are similar to verb cases. There are domain-specific criteria for selecting a range of semantic roles. In this domain the semantic roles include: **agent**, **instigator**, **experiencer**, **instrument**, **theme**, **object1**, **object2**, **location**, **actor**, **patient**, **source**, **reference_pt** and **goal**. Semantic roles can be filled either by a syntactic con-

stituent supplied by a mapping rule or by reference resolution, requiring close co-operation between semantics and pragmatics. A proposed role filler must satisfy the semantic class restrictions on the role. In order to control how roles are filled, certain semantic roles are categorized as OBLIGATORY, indicating that they must be filled by a syntactic constituent. Other roles, in the context of particular verbs, are categorized as ESSENTIAL, which signals pragmatics to fill the role even if there is no syntactic constituent available. The default categorization is NON-ESSENTIAL, meaning the role does not need to be filled. These classifications are explained in more detail with extensive examples in [23] and [24], included as Appendices B and K respectively. The clause analysis algorithm described in detail below guides the semantic interpreter in filling the semantic roles of verb decompositions. There are subtle but interesting differences in the algorithms for semantic interpretation of the other predicating expressions which are explained in more detail in the next section.

The mapping rules that are used to fill roles with syntactic constituents can be illustrated using the verb *replace* as shown in Figure 4. The decomposition of *replace* indicates that an **agent** can use an **instrument** to exchange two **objects**. The **agent** mapping rule specifies that the **agent** can be indicated by the subject of the clause.

The mapping rules make use of intuitions about syntactic cues for indicating semantic roles first embodied in the notion of case [10, 21]. Some of these cues are quite general, while other cues are verb-specific. The mapping

DECOMPOSITION:

```
replace ←
        cause(agent(A),
          use(instrument(I),
            become(exchange(object1(O1),object2(O2)))))
```

MAPPING RULES:

```
agent(A) ← subject(A) / X

object1(Part1) ← obj(Part1)/ cause(agent(A),Repair_event)

object2(Part2) ←
     pp(with,Part2) /
     cause(agent(A),use(I,exchange(object1(O1),object2(Part2))))
```

Figure 4.
Partial Lexical Semantics for *replace*

rules can take advantage of generalities like "SUBJECT to AGENT" syntactic cues while still preserving verb-specific context sensitivities when necessary. This is accomplished by making the application of the mapping rules verb-specific through the use of optional PREDICATE ENVIRONMENTS. The previous rule is general and can be applied to every **agent** semantic role in this domain as indicated by the uninstantiated variable X on the right hand side of the "/", i.e., the predicate environment of the **agent**, which in this case can be anything. Other rules, such as "WITH-PP to OBJECT2," are much less general, and can only apply under a set of specific circumstances.

In general it can be assumed that syntactically different types of predicating expressions require different sets of mapping rules. For example, the mapping rules for the nominalization *replacement* will not be the same as the ones just described, although the decomposition will be identical.

The semantic roles also have semantic class restrictions, which vary more from verb to verb than the mapping rules do, although there are still some general ones. The general semantic class restriction on an **agent** is that it must be animate, and an **instrument** for *repair* verbs must be a tool. The semantic class restrictions on the two **objects** of *replace* are more complicated, and quite verb-specific. They must be machine parts, have the same type, and yet also be distinct objects. In addition, the **object1** must already be associated with something else in a **haspart** relationship, in other words it must already be included in an existing assembly. The opposite must be true of **object2**: it must not already be included in an assembly, so it must not be associated with anything else in a **haspart** relationship. (In this domain one can only replace something that is a part of a whole.) Thus the procedures which check semantic class restrictions must have access not only to the domain model, but also to the current discourse context The semantic class restrictions are characteristic of lexical items independent of their syntactic function.

For the verb usage of *replace*, both **object1** and **object2** are ESSENTIAL semantic roles. Whether or not they are mentioned explicitly in the sentence, they must be filled. If they are not mentioned explicitly, reference resolution

will fill in the role, preferably by an an entity that has already been mentioned. If there is no suitable previously mentioned entity, one will be created [20]. This is not done for non-essential roles, such as the **agent** and the **instrument** in the same verb decomposition, which are simply left unfilled if not mentioned. The **instrument** is rarely mentioned, and the **agent** could easily be left out, as in *The disk drive was replaced at 0800.*[6] In other domains, the **agent** might be classified as obligatory, and then it would have to be filled in.

The typing of a role as OBLIGATORY or ESSENTIAL is used mainly by the clause-analysis algorithm, as explained in Section 9.3. The question of how this typing ports to other predicating expressions is currently under investigation.

8. Designing the CASREP Lexical Semantics

The previous section has described the basic algorithm used by PUNDIT. This section describes the domain model and the lexical semantics that act as input to that component. The domain model currently has to be designed from scratch; we hope that current research on common sense reasoning will begin to provide portable components for new domains. The lexical semantics, on the other hand, can often borrow mapping rules and predicate decompositions from prior domains. One of our goals has been to create tools that can speed up the task of designing the lexical semantics (see Section 13).

[6]Note that an elided subject is handled quite differently, as in *replaced disk drive*. The missing subject is assumed to fill the **agent** role, and an appropriate referent is found by reference resolution (see Section 10).

8.1. The CASREPs Domain Model

The long-term goal is to incorporate NYU's sophisticated model of a starting air compressor into the PUNDIT domain model. This has been delayed because of difficulties interfacing Prolog and CommonLisp. Meanwhile, PUNDIT has made use of a rudimentary semantic net domain model of a starting air compressor that was built using three basic predicates: **system**, **isa**, and **haspart**. For example, an instance of the system (a starting system diesel generator) is indicated by **system(ssdg2)**. The **isa** predicate associates TYPES with COMPONENTS, such as **isa(diesel,engine)**.

This method of representation results in a general description of a starting air compressor. Specific compressors represent INSTANCES of this general representation. When a particular report is being processed, **id** relations are created by noun phrase semantics to associate the specific component parts being mentioned with the part descriptions from the general machine representation. So a particular sac, i.e., a starting air compressor and its parts, would be indicated by predicates such as these: **id(sac,sac1)**, **id(pump,pump1)**, **haspart(ssdg1,sac1)**, **haspart(sac1,pump1)**, etc.

The messages follow a standard format, beginning with fixed-field information specifying the ship, its base and location, and the subject of the report, e.g., the particular piece of equipment. The messages we have analyzed are concerned with starting air compressors. The final section of the message is the REMARKS section, which is for free text. This is the section that serves as input

to our system. It generally mentions the specific problem first, and then recounts any tests that have been performed to find out more information, and the results, if any, of the tests.

8.2. The CASREP Lexical Semantics

The lexical semantics for the CASREP domain requires several features that were not required by earlier domains [23]. A much larger range of verbs including aspectual and abstract verbs is involved as well as complex intra-sentential time relationships. These necessitate a more sophisticated system of verb classification as well as a theory of time analysis. It is interesting to note that these extensions were incorporated without disturbing the core distinguishing features of the lexical semantics in the original implementation, as outlined below.

8.2.1. Distinguishing Features

The first application of the semantic analyzer used by PUNDIT was the pulley domain of the Edinburgh Mecho project [2], which consists of a corpus of pulley word problems from physics text books. The goal of the project was to produce a semantic representation of the problem, derive equations to model the representation, and then solve for variables in the equations.

The pulley problems form a very limited domain, with only 31 verbs and a small set of domain-specific semantic roles which include **agent**, **intermediary**

(a type of instrument), **object1**, **object2** (types of patients), **locpt**, **loc**, and **time.**[7] The verbs fall into five main categories, all of which are fairly concrete: **contact verbs**, **support verbs**, **location verbs**, **motion verbs**, (including **cause_motion** verbs), and **quantity verbs**. There are three distinguishing features of the pulley domain semantics that ported consistently to the CASREP domain:

1) The uniformity of the semantic roles of each verb category. For instance, the **location verbs** all require an **object1** semantic role and a **loc** semantic role.

2) The generalization of the mapping rules for semantic roles that occur in more than one category. For example, an **object1** can always be the object of the sentence no matter which verb decomposition it is figuring in.

3) The generality of the verb decompositions. For example, the verbs whose decompositions include optional **agents** and/or **intermediaries** can decompose in different ways, depending on the presence or absence of the optional role. This allows one general decomposition to be used to produce several different final representations each of which is dynamically tailored to fit the context of a particular verb usage.

[7]The treatment of time as a semantic role was simply a temporary device until a more general treatment could be developed.

8.2.2. CASREP Verb Hierarchy

The CASREP corpus involves 92 verbs, which fall into the broad categories of ASPECTUAL, COMPLEX and BASIC, each of which has several verb subcategories. The seven BASIC subcategories correspond closely to the verb subcategories defined for the pulley corpus, with at least one verb argument being a concrete object that is a component of the system being discussed, such as *diesel was operating.* The COMPLEX verbs can take entire propositions or clauses as arguments, as in *investigation revealed contamination.* The ASPECTUAL verbs can also take propositions as arguments, and simply add time information to the representation of the proposition. The tree in Figure 5 gives the three basic categories with their respective subcategories. The taxonomy in Appendix F lists the verbs in each subcategory for the CASREP corpus.

In addition to its function in determining the argument structure of verb, the verb taxonomy is central to the component which creates database entries, described in Section 12.2. For example, the database structure includes relations for storing information about damage, normal operations observed, and symptoms of abnormal conditions, among others. In order for the database component to determine in which relations the information from the current sentence is to be stored, the verb hierarchy is consulted. For example, verbs of the class **operatingP**, such as *start, operate* and *rotate,* map to the relation **normal_operation**.

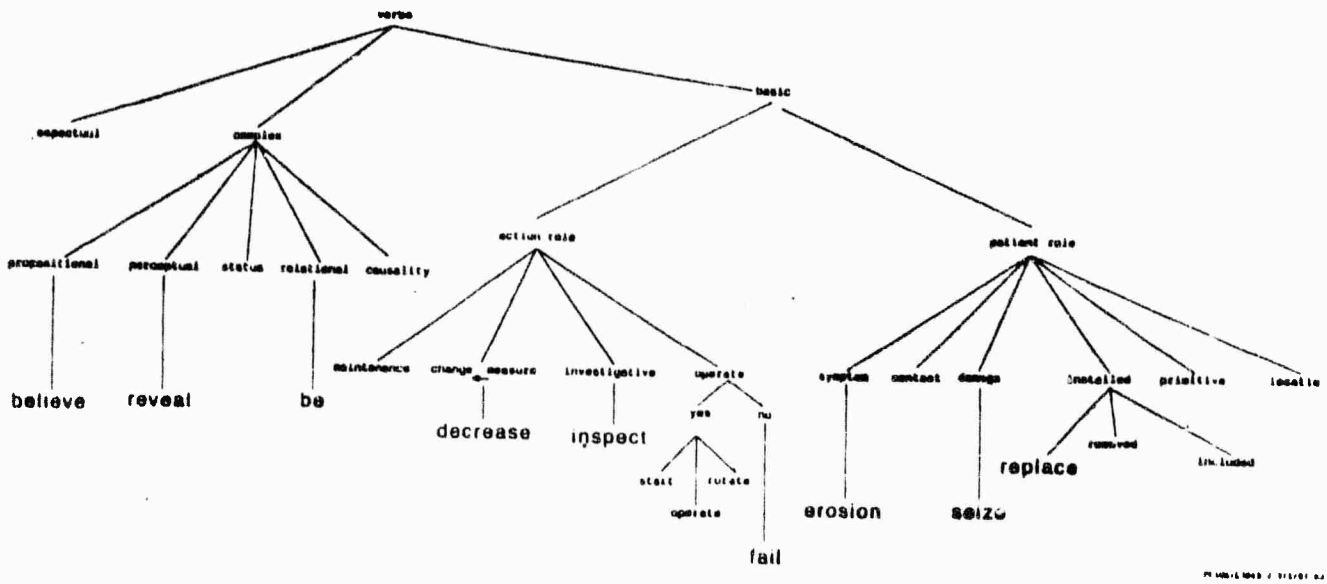

Figure 5.
Verb Hierarchy

8.2.3. CASREP Semantic Roles

The claim with respect to semantic roles is that they capture generalizations about the relationships between syntactic constituents and the predicate-argument representations of verbs. This claim is supported by the development of the CASREP lexical semantics, since the set of semantic roles did not grow in proportion to the set of verbs. There are three times as many verbs (92 to 31) in the CASREP corpus as there were in the pulley problem corpus, but only about half as many semantic roles (11 to 7). However, the much larger set of verb subcategories involving each semantic role did allow for greater regularization. It is encouraging to note that the CASREP semantic roles correspond more closely to traditional linguistic thematic roles, and appear to be much less domain-specific than roles used in earlier domains. This has allowed us to take advantage of current literature on regularities in syntactic realizations of verb argument structures, such as the recent results of the MIT lexicon project [31, 19]. Some of these regularities can be expressed as tests for determining appropriate semantic role classifications of predicate arguments. For instance, *metalb* in *metal contamination of the oil* can be classified as an **instrument** since it can also be expressed as a *with-* or *by*-prepositional phrase as in *the oil was contaminated with metal* or *the oil was contaminated by metal.* This is discussed in more detail in Appendix I. An important area for future research is the incorporation of these tests into a tool for aiding the design of lexical semantics for new domains.

8.2.4. Incorporating Aspectual Information

The time-analysis algorithm makes use of the inherent temporal class that a verb falls into, i.e., state, event or process, as discussed in [26] and [27] (also included as Appendices J and L). These classes are reflected in both the verb decompositions and the choice of semantic roles. The **becomeP** operator that is included in many verb decompositions indicates the **event** temporal class. A decomposition that includes an **actor** semantic role indicates the **process** temporal class. The particular usage of a verb can alter its temporal class, as explained in Section 11. The final representation of the verb is either a representation of an **event**, a **state**, or a **process**. These are all considered to be discourse entities, and can be referred to in the rest of the text. In this way nominalizations and pronouns can be used to refer to previously mentioned events, as in the following discourse.

sac failed.
failure occurred when diesel was operating.
investigation [of failure] revealed sheared drive shaft.

The final deep semantic representation is called an INTEGRATED DISCOURSE REPRESENTATION, or IDR, since it includes everything mentioned in the discourse, whether concrete object, animate object, event, state or process, and further references to any of these types of entities can be accurately detected by reference resolution, as explained in Section 10.

The third distinguishing feature of the original theory of lexical semantics was the dynamic generation of final representations. The form of the representation could vary from the decomposition rule, depending on the context of the verb usage. For example, the explicit mention of the optional intermediary *string* in *two particles are connected by a string* causes a more complex final representation to be created than is created for *two particles are connected to each other.* This feature still holds, but the representations are now dependent on aspectual information as well as the absence or presence of optional semantic roles. For example, the tense of *sac is failing* causes it to be represented as a **process** while *sac failed* is represented as an **event** (see Section 11).

9. Interaction Between Syntax, Semantics and Pragmatics

The analysis algorithm described in Section 7 has been extended to cover predicating expressions in a full range of syntactic environments, including noun phrases and modifiers as well as verbs, as described in Section 9.2. One implementation of the algorithm is used to process all of these types of predicating expressions. This is done by having the interpreter operate in a different mode for each different type of syntactic environment, as described in Section 9.3. An essential contributor to this modularity is the ISR produced by the syntactic parser. The ISR regularizes the parse information from the different syntactic environments as much as possible in order to simplify the operation of all the different modes of the semantic analysis interpreter. For example, the ISR finds

the subject of a non-finite clause and labels it as such, thus providing additional input for the mapping rules. However, it does not change the syntactic typing of a nominalized verb as a noun phrase, and it is up to semantics to recognize nominalizations and deal with them appropriately.

Since each syntactic environment containing a predicating expression can have embedded within it another syntactic environment containing another predicating expression, the semantic analysis algorithm must be recursive in the same way that the syntactic parser is. For example, in the analysis of *Investigation of decreased pressure revealed metal contamination in oil*, the initial call to semantic analysis pertains to the analysis of the *reveal* clause. This in turn requires the analysis of the *investigation* phrase, a nominalization, which is the proposed filler for one of the semantic roles of *reveal*. The analysis of *investigation* requires the analysis of the noun predicate *pressure*, as the head noun, and a proposed filler for one of *investigation*'s semantic roles. The modifier of *pressure* is *decreased*, the participial form of the verb *decrease* which is being used as an adjective. This is also a predicating expression which has to be analyzed. After successfully completing the analysis of each of these phrases, the interpreter will proceed to analyze the other nominalization, *contamination*, the proposed filler for the second semantic role of *reveal*, and another predicating expression, and so on. In all of these levels of recursion, the relevant time information and current discourse context must be kept straight so that the final representation is accurate. This requires a carefully integrated control struc-

ture for the semantic and pragmatic components.

The two basic paradigms used for interaction between the different components are 1) "generate and test" and 2) mutual recursion. The specifics of the interaction are described in detail below. The tight integration of control has been particularly effective in the achievement of two of the original research goals: 1) the filling in of implicit information, and 2) the recovery of information from incomplete sentences. Both of these tasks are described in detail in Section 9.4, and Appendix B.

9.1. An Integrated Control Structure for Semantics and Pragmatics

The two pragmatic components that currently interact with semantic analysis are reference resolution (see Section 10) and time analysis (see Section 11). By confining interaction to well-defined points between the separate components, the interaction can be carefully controlled. In the current implementation, the interaction is not used to reject the semantic analysis, but simply to add more information to it. Interaction with reference resolution is used to provide specific referents for the noun phrases that are proposed fillers for semantic roles, or for implicit fillers, as explained below. Interaction with time analysis is used to produce the Integrated Discourse Representation, which includes the status of the discourse entity (**event**, **state** or **process**) being represented. This may also involve completing the semantic analysis of temporal sentence adjuncts that were not directly relevant to the main predicating expression and

had not yet been analyzed.

The primary mode of interaction between semantics and reference resolution is the "generate and test" paradigm, while for semantics and time analysis, it is mutual recursion. Our initial experiments with the interaction between syntax and semantics have suggested that the "generate and test" paradigm may have distinct advantages over more complicated modes of interaction with respect to both modularity and efficiency. We have considered several alternatives for both syntax/semantics interaction and semantics/reference resolution interaction. In particular, we are examining in detail the possible advantages of having semantics play a more instructional role with respect to the parser (see Appendix N). Semantics would analyze a partial parse, and then make suggestions about likely successive constituents. The parser would order its grammar rules accordingly, so that it would try to parse these expected constituents next. However, this involves a complicated interaction which may have a detrimental effect on modularity and efficiency. The parser has to keep track of the local discourse context (for reference resolution), and in any highly predictive version of this mechanism the semantic analyzer must apply every possible mapping between syntactic constituents and thematic roles that might yield an appropriate argument. It also involves running the full semantic analysis at several points. This may result in a negative trade-off in terms of execution time, since the added semantic processing outweighs the reduced synactic parsing. By restricting semantics to a strictly selectional role, i.e., the "generate and test"

paradigm, modularity can be preserved along with possible improvements in parsing efficiency. We are still examining this issue, but are impressed by the potential efficiency advantages offered by a selectional mode of interaction rather than an instructional mode. It is interesting to note that other researchers are currently expressing similar reservations with respect to the instructional mode. [talk by Henry Thompson, April, U of Penn] and [Mitch Marcus, personal communication].

We have not yet been able to test this hypothesis with respect to the interaction between semantics and pragmatics, since we have not yet attempted to use pragmatics to help disambiguate semantic analyses. Contextual information could be helpful in determining the appropriate definition of polysemous items such as *loss* (e.g., *loss of pressure* vs. *loss of sac*). It will be interesting to see if the "generate and test" paradigm offers advantages for this type of interaction as well.

9.1.1. Interaction with Reference Resolution

The points of interaction between semantic analysis and reference resolution are:

1) Noun phrase semantics calls reference resolution after calling semantic analysis for each predicating expression that is part of a noun phrase.

2) Semantic analysis can also call reference resolution directly in order to fill in a semantic role not filled by a syntactic

constituent.

Reference resolution creates labels for entities when they are first directly referred to, or when their existence is implied by the text, and then recognizes subsequent references to the same entities (see Section 10). Reference resolution is traditionally performed after the semantic analysis of a clause is complete. In contrast, we perform the semantic and pragmatic analysis of noun phrases during semantic analysis of the predicating expression associated with the verb, when the semantic roles are being filled. This allows the semantic class restriction information on a semantic role to be considered in determining the referent of the noun phrase. This is especially helpful in determining referents of noun phrases that are not fully described, such as pronouns and elided consituents (see Section 9.4).

9.1.2. Interaction with Time Analysis

The points of interaction between semantic analysis and time analysis are:

1) Semantic analysis calls time analysis after the semantic analysis of most, but not all, predicating expressions.

2) Time analysis can in turn call semantic analysis for the analysis of time adverbials.

Time analysis is performed after the semantic analysis of most predicating expressions. In general, time analysis expects the ISR of the expression to contain any tense and aspect information available from the parse. The time analysis algorithm will also look for certain adverbial phrases (e.g., prepositional

phrases or subordinate clauses) that could add additional time information (see Section 11). These phrases must be given a semantic analysis which will in turn occasion another call to time analysis. Therefore time analysis and semantic analysis have to be mutually recursive. For example, in analyzing the sentence *failure occurred during engine start*, the prepositional phrase *during engine start* is a time adverbial rather than an argument of the verb *fail*. Semantic analysis will first produce an analysis for *failure occurred*, and leave the time adverbial unanalyzed. Time analysis is then called, and it will process the unanalyzed prepositional phrase. Noun phrase semantics will be called to analyze the noun phrase *engine start*. Since *start* is a nominalization, this will occasion another call to semantic analysis, which will in turn call time analysis again. In the end the *failure* event and the *starting* event will both have been given time reference points which will have been related to each other.

9.2. Adapting the Analysis Process to New Predicating Expressions

The general task of the semantic analyzer is the analysis of predicating expressions. Predicating expressions occur in a full range of syntactic environments in the CASREPs, and each receives a different representation from the ISR. For each of these predicating expressions, adjustments must be made to the semantic and pragmatic components to insure the production of an accurate final representation for the IDR. This representation must include the instantiated semantic decomposition, the correct resolution of referents, and an accu-

rate analysis of intra-sentential time relations. In general, we have two basic modes for the operation of semantic analysis: 1) the analysis of clauses and 2) the analysis of noun phrases. As expected, there are major differences in the syntactic structure of these phrases, as well as pragmatic expectations with respect to reference resolution and time analysis. In practice, there are many types of predicating expressions that do not behave semantically as either simple clauses or simple noun phrases. There are verbs that can be used as nouns, e.g., nominalizations such as *failure* and *monitoring*, and nouns that can be used as verbs, e.g., deverbal nouns such as *crack*. Verb participles can be used as noun modifiers, and nouns can be used to modify other nouns. Each new type of mixed-category predicating expression requires a customized version of the semantic analysis algorithm that makes allowances for its particular requirements. For instance, nominalizations and non-finite subordinate clauses need to go through time analysis, even though they do not have syntactically marked tense. By allowing the tense of the matrix clause to be passed around as a parameter during the recursive calls to semantic analysis, time analysis can be given the information it needs to produce an accurate final representation.

The next section describes in detail the differences in the analyses of nominalizations and main clauses, focusing on the necessary changes to the interaction between semantics and pragmatics. There is an additional special provision that has to be made for nominalizations that is concerned with the input from the ISR. There are two types of nominalizations, 1) nominalizations which

are formed derivationally, such as *failure*, and 2) nominalizations which are formed productively, such as *monitoring*. In order to pass the same input to semantic analysis, so that nominalizations can receive a uniform treatment, each type must be handled slightly differently. The derivational forms are handled with individual rules, and the productive forms are handled by a general rule. The ISR has to mark the productive form as a GERUND, the *-ing* form of a verb being used as a noun, so that the next component knows to apply the general rule. It is only at the point when the ISR is being formed that this information is available, so it must be analyzed there, and passed along in a form that is meaningful to the next component.

Figure 6 summarizes the different types of predicating expressions that the system handles: the entries marked by a single star are currently being worked on and should be completed by the time this report is circulated; approaches for the entries marked by double stars have not yet been developed, and will require an extension of the IDR.

The fact that we have been able to extend the system as widely as we have without changing the basic structure is strong support for our claims to modularity and portability. Relative clauses have not yet been included in the range of syntactic environments, since they did not occur in the CASREP corpus. Neither have we attempted to deal with modality, negation and quantification, none of which occur with great frequency in the CASREP corpus. Each of these will have to be carefully integrated with the theories of lexical semantics and

Verb Phrases

main clause verbs
subordinate clause verbs
non-finite subordinate verbs
Conjoined verbs
*Polysemous verbs

Noun Phrases

nominalizations
noun predicates
Conjoined noun phrases
*Polysemous noun phrases

Prepositional Phrases

prepositional phrases attached to nouns
prepositional phrases attached to verbs
**prepositional phrases attached to assertions

Adjectival Modifiers

adjectives
nouns that are left modifiers
*participles that are left modifiers
**clauses that are modifiers

Adverbial Modifiers

time adverbs modifying assertions
*goal adverbs modifying assertions
**other adverbs modifying assertions, such as manner adverbs

Figure 6.
The Range of Syntactic Environments for Predicating Expressions

discourse representation.

9.3. Operating the Interpreter in Different Modes

The original semantic analysis algorithm for Inference-Driven Semantic Analysis was designed for clause analysis based on decompositions representing a verb's predicate-argument structure. That algorithm extended readily to the analysis of other predicating expressions such as nominalizations and noun predicates. There are important differences associated with each type of predicating expression, but the basic approach has stayed constant. This consists of decomposing an expression into its component semantic predicates, and then filling in arguments either with syntactic constituents or by pragmatic deduction. All of the predicating expression outlined above are executed by one meta-level implementation of the semantic analysis interpreter which can operate in any one of several modes. The mode determines which optional steps in the algorithm will be performed, such as time analysis or updating of the discourse structure. The mode can also determine which set of syntactic mapping rules is relevant, and whether or not unfilled obligatory roles should cause failure. This can best be illustrated by examining the differences between the clause analysis algorithm and the nominalization algorithm in more detail. (See [24], included as Appendix K, for a detailed example.)

The clause-analysis algorithm begins by decomposing the verb. Then one pass is made through the semantic roles, attempting to fill each role in turn.

First, syntactic constituents are examined to fill the role, and if an appropriate constituent is found that satisfies the semantic class restrictions, then the role is filled, and the next role can be considered. If there are no appropriate syntactic constituents and the role is obligatory, failure results immediately. On the other hand, if there are no syntactic constituents and the role is essential, reference resolution is called to produce a filler. Finally, if there are no syntactic constituents, but it is a non-essential, non-obligatory role, the role is simply left unfilled.

Nominalizations are processed very similarly to clauses, but with a few crucial differences, both in linguistic information accessed and in the control of the algorithm. With respect to the linguistic information, the decomposition of a nominalization is the same as that of its corresponding verb, but the mapping rules differ, since syntactically, a nominalization is a noun phrase. For example, where a likely filler for the **patient** of the verb *fail* is the syntactic subject, a likely filler for the **patient** of the related nominalization *failure* is an *of*-prepositional phrase. In addition, noun phrase modifiers are not syntactically obligatory.[8] Secondly, because nominalizations may themselves be anaphoric, there are two separate role-filling stages in the algorithm instead of just one. The first pass is for filling roles which are explicitly given syntactically; essential roles are left unfilled. This is because if a nominalization is being used anaphorically, some of its roles may have been specified or otherwise

[8] This suggests the hypothesis that OBLIGATORY roles for clause decompositions automatically become ESSENTIAL roles for nominalization decompositions. This hypothesis seems to hold in the current domain; however, it will have to

filled when the event was first described. The anaphoric reference to the event, the nominalization, would automatically inherit all of these role fillers, as a by-product of reference resolution. After the first pass, the interpreter looks for a referent, which, if found, will unify with the nominalization representation, sharing variable bindings. This is a method of filling unfilled roles pragmatically that is not currently available to clause analysis.[9] However, it is important to fill roles with any explicit syntactic arguments of the nominalization before attempting to resolve its reference, since there may be more than one event in the context which a nominalization could refer to. For example, *failure of pump* and *failure of sac* can only be distinguished by the filler of the **patient** role. After reference resolution, a second role-filling pass is made where still unfilled roles may be filled pragmatically with default values.

9.4. Making implicit information explicit

This section describes how the integrated control of the separate components has been effective in solving the problem of recovering implicit information. We have isolated two types of implicit entities: syntactic entities (i.e., missing syntactic constituents), and semantic entities (i.e., unfilled semantic roles). The key is allowing syntax and semantics to recognize the missing linguistic entities as implicit entities, so that they can be labelled as such, which

be tested on other domains. We are indebted to James Allen for this observation.

[9] Clauses can be anaphoric, as discussed in [6]. In order to handle cases like these, something analogous to refer-

in turn allows reference resolution to find specific referents for the entities. In this way the task of making implicit linguistic information explicit becomes a subset of the tasks performed by reference resolution. Reference resolution uses different methods for filling in implicit syntactic and semantic entities, but both methods are part of the standard resolution component, and are used for general noun phrase reference problems.

In order to recognize implicit entities, both syntax and semantics must distinguish between optional and obligatory entities. If syntactically obligatory entities are missing from the parse, as they are in sentence fragments (see Section 5.3), then they are assumed to be implicit. A necessary semantic role is an ESSENTIAL role, and if it is not mentioned explicitly it is also assumed to be implicit. The assignment of syntactic constituents to optional and obligatory categories is quite general, and would port easily from domain to domain. The assignment of semantic roles to optional and obligatory categories is very domain specific, and the same verb could have its semantic roles assigned differently in different domains. For example, adverbial prepositional phrases are optional syntactic entities. Their absence in the parse of a clause is no cause for concern. But the absence of a subject, clearly an obligatory entity, is another matter entirely. The parser must recognize that the word string is a syntactically correct sentence fragment with an *elided* subject. Then semantics can assign a likely semantic role to the subject, and reference resolution can be

ence resolution for clauses may be required. However, a treatment of this has not yet been implemented in PUNDIT.

called to find a referent. At this point reference resolution knows the syntactic constituent and the semantic class restrictions on the semantic role, and the reference problem can be treated very similarly to a pronoun reference problem as discussed in Section 10. This allows PUNDIT to fill in *ship's force* as the **agent** of *believed coupling from diesel to pump to be sheared.*

Obligatory semantic roles (ESSENTIAL roles) are treated somewhat differently. In this domain, the verb *replace* has two optional semantic roles, the agent and the instrument, and two essential roles, the old object being replaced and the new object replacing it. If semantics recognizes that there are no syntactic constituents available to fill an essential semantic role, then a special call to reference resolution is made. The assumption is that the filler of the role must be obvious from the context, and reference resolution is again given the semantic class restriction information. This is more like finding the referent of a noun phrase with no article, where the type is known, but the item may or may not have already been mentioned explicitly. If it has not been mentioned, then reference resolution will create an entity of the appropriate type. The absence of a syntactic constituent for an optional semantic role is of no moment, and the semantic role is simply left as a variable.

This close integration of syntax, semantics, and pragmatics enables PUNDIT to correctly supply the missing information in the type of sentence fragments illustrated by Figure 7.

Example:	Semantic Processing	Result
tvo: *'Request replacement of SAC.'*	Create a subject and treat as pronoun	[ship's force] *request replacement of SAC.*
zerocopula: *'Exact cause of failure unknown.'*	Replace missing verb with *'be'*	*Exact cause of failure* [be] *unknown*
nstg_frag: *'Loss of lube oil pressure.'*	Insert verb *'occur'* or *'exist'*	*Loss of lube oil pressure* [occur].
predicate: *'Beyond shipboard repair'*	Create pronoun subject (as for tvo) and insert *'be'*	[SAC] [be] *beyond shipboard repair.*
null determiner: *'Diesel was operating.'*	First try definite, then try indefinite (see reference resolution)	[a] *diesel was operating.*

CG200/A3217b-4 4/30/88

Figure 7.
Recovering Implicit Information

10. Reference Resolution

When the semantic interpreter is ready to fill a semantic role, it calls **constituentAnalysis.** This component can either call a component to analyze noun phrases, or can recursively call the semantic interpreter, if the proposed filler of the semantic role is sentential. Noun phrase analysis consists of two components: noun phrase semantics, discussed above in connection with the semantic interpreter, and reference resolution. The function of reference resolution is to propose a referent for the constituent associated with the semantic role to be filled. For example, if the verb is *replace* and the semantic interpreter is filling the role of **agent** with the surface syntactic subject, reference resolution would be called for the subject. After a proposed referent is chosen, any specific selectional restrictions on the agent of *replace* (such as the constraint that the agent has to be a human being) are checked. If the proposed referent fails selection, backtracking into reference resolution occurs and another referent is selected. Cooperation between reference resolution and the semantic interpreter is discussed in detail in [23], also included as Appendix B.

A number of different types of noun phrases are handled by the reference resolution component in PUNDIT. These inciude definite and indefinite noun phrases, as well as noun phrases without determiners; pronouns and elided noun phrases; certain types of *one*-anaphora; and certain types of non-specific references, such as *motor* in *Ordered a motor.* The system also handles references to situations as well as references to objects. The discussion of reference resolu-

tion is divided into two sections. The first section describes our treatment of full noun phrases, and the second is concerned with the uses of focusing to handle various cases of reduced reference, such as pronouns, elided noun phrases, and *one*-anaphora.

10.1. Full Noun Phrases

The definite determiner *the* typically signals that a referent has been previously mentioned or is otherwise known, so that *the pump* is taken to refer to some known pump. Conversely, *a pump* is taken to refer to a pump that is not previously known to the reader. This distinction can be used by a reference resolution component to decide whether or not the text is mentioning a new entity. However, in the style of speech characteristic of maintenance reports such as the CASREPs, determiners are nearly always omitted. Their function must therefore be replaced by other mechanisms. One possible approach to this problem might be to have the system try to determine what the determiner would have been, had there been one, insert it, and then resume processing as if the determiner had been there all along. This approach was rejected for two reasons. The first is that it was judged to be more error-prone than simply equipping the reference resolution component with the ability to handle noun phrases without determiners directly. The second reason for not selecting this approach is that this would eliminate the distinction between noun phrases which originally had a determiner and those which did not. At some point in the development of the system it may become necessary to use this information.

The current approach deals with this issue as follows. If a noun phrase has an explicit indefinite determiner (and the noun is not *dependent*, in the sense discussed in the next section), the noun phrase is assumed to refer to something not previously known, and a new unique identifier is created for that referent. In contrast, other noun phrases trigger a search through the discourse context for a previously mentioned referent. Both noun phrases with an explicit definite determiner, and those with no determiner are handled in this way. This amounts to making the assumption that noun phrases without determiners are definite. If the search for a previously mentioned referent fails, then another search determines if the referent of the noun phrase is related to some other referent in focus (that is, whether the two referents are *implicit associates* as discussed below). If this fails, then a new unique identifier is created, just as if the noun phrase had been indefinite.

In order to determine whether a referent has been previously mentioned, the properties in the current noun phrase are matched against the properties of previous noun phrases. As long as all of the new properties are also properties of the old referent, the referents match. So in a discourse such as *New sac arrived. Sac was installed.* the second time the sac is mentioned, it does not again have to be referred to as being new. It is also possible to refer to an entity for the second time using a superordinate term, as in *Sac failed. The machine has been failing repeatedly.* PUNDIT is able to recognize that *machine* is a reference to the sac.

10.2. Uses of Focusing

A focusing algorithm based on surface syntactic constituents is used in the processing of several different types of reduced reference: definite pronouns, *one*-anaphora, elided noun phrases, and implicit associates. The focusing mechanism in this system consists of two parts--a **FocusList**, which is a list of entities in the order in which they are to be considered as foci, and a focusing algorithm, which orders the **FocusList**. A detailed description of the focusing mechanism is available in [5], also included in this report as Appendix C.

Pronoun resolution is done by instantiating the referent of the pronoun to the first member of the **FocusList.** Selectional restrictions on referents are checked after reference resolution is exited. Backtracking into reference resolution occurs if selection fails and then then next member of the **FocusList** is proposed as a referent. If all focused entities are rejected, a domain-specific default is used.

The reference resolution situation in the maintenance texts is, however, complicated by the fact that there are very few overt pronouns. Rather, in contexts where a noun phrase would be expected, there is often elision, or a zero-NP as in *Received new sac* and *Investigated failure.* Zeroes are handled as if they were pronouns. That is, they are assumed to refer to the focus. The hypothesis that elided noun phrases can be treated in the same way as pronouns is consistent with previous claims in [12] and [17] that in languages such as Russian and Japanese, which regularly allow zero-NP's, the zero corresponds

to the focus. If these claims are correct, it is not surprising that in a sub-language like that found in the maintenance texts, which also allows zero-NP's, the zero should correspond to the focus.

Focusing is also used in the processing of certain full noun phrases, both definite and indefinite, which involve *implicit associates*. The term implicit associates refers to the relationship between *a disk drive* and *the motor* in examples like *The field engineer installed a disk drive. The motor failed.* It is natural for a human reader to infer that the motor is part of the disk drive. In order to capture this intuition, it is necessary for the system to relate the motor to the disk drive of which it is part. Relationships of this kind have been extensively discussed in the literature on definite reference. For example, implicit associates correspond to *inferrable* entities described in [30], the *associated use definites* of [13], and the *associated* type of implicit backwards specification discussed in [35]. Sidner suggests that implicit associates should be found among the entities in focus. Thus, when the system encounters a definite noun phrase mentioned for the first time, it examines the members of the **FocusList** to determine if one of them is a possible associate of the current noun phrase. The specific association relationships (such as **part-whole**, **object-property**, and so on) are defined in the knowledge base.

This approach is also used in the processing of certain indefinite noun phrases. In every domain, there are certain types of entities which can be classified as *dependent*. By this is meant an entity which is not typically men-

tioned on its own, but which is referred to in connection with another entity, on which it is dependent. For example, in the starting air compressor domain, *oil* is classified as such a dependent entity, since it is normally mentioned with reference to something else, such as a *starting air compressor.*

PUNDIT extends focusing to the analysis of *one*-anaphora following [4], which claims that focus is central to the interpretation of *one*-anaphora. Specifically, the referent of a *one*-anaphoric noun phrase (e.g., *the new one, some defective ones*) is claimed to be a member or members of a set which is the focus of the current clause. For example, in *Three sacs installed. One failed*, the set of three sacs is assumed to be the focus of *One failed*, and the sac that failed is a member of that set. The main computational advantage of treating *one*-anaphora as a discourse problem is that the basic anaphora mechanism then requires little modification in order to handle *one*-anaphora. Alternatively, treating *one*-anaphora as a purely syntactic phenomenon would require an entirely separate mechanism.

The data structures that retain information from sentence to sentence in the PUNDIT system are the **FocusList** and the **CurrentContext**. The **FocusList** is a list of all the discourse entities which are eligible to be considered as foci, listed in the order in which they are to be considered. Events as well as objects which have been mentioned are included in the **FocusList**. The **CurrentContext** contains the information that has been conveyed by the discourse so far. The **CurrentContext** would contain three types of informa-

tion:

(1) **Discourse** ids, which represent classifications of entities, such as **id(pump,[pump1]).**

(2) Facts about part-whole relationships (**hasparts**), such as **haspart([sac1],[motor1]).**

(3) Representations of the situations in the discourse, such as the following representations for *Sac failed*, which represent the failure event and the state which results from it:

event([fail1],become(inoperative(patient([sac1]))),moment([fail1]))

state([fail2],inoperative(patient([sac1])),period([fail2]))

Recent work [3f] (also included as Appendix M) introduces the notion of a TEMPORAL FOCUS, and suggests that the interpretation of temporal information in texts makes use of a focusing mechanism very similar to that used for reference resolution. We plan to generalize the current focusing mechanism to include procedures for handling temporal focus by incorporating insights from this work.

11. Temporal Analysis in the CASREPs Domain

11.1. Overview

The CASREPs domain has turned out to be an appropriate one for implementing a temporal component to analyze the time information contained expli-

citly within the individual sentences of a text. Within one sentence, several different situations may be mentioned, linked together by explicit temporal connectives such as *before* and *after*. Because the texts have a simple rhetorical structure, a large amount of temporal information can be extracted from the temporal semantics of the sentence, even though in the current implementation, inter-sentential temporal relations are not yet handled (this is one of the extensions planned for the follow-on work). However, the implementation of the temporal semantics component in PUNDIT lays the necessary groundwork for the development of inter-sentential temporal relations along lines proposed in Webber [37].

In natural language, time can be associated with explicit means of measuring time, such as clocks and calendars, but is more often specified relative to the time of other events or situations. Different types of situations have different ways of evolving through time that affect how a language understander evaluates what is asserted to have happened or to be taking place in the world. PUNDIT's temporal component dynamically computes three types of temporal structure associated with three distinct types of real-world situations--**states**, **processes** and **transition events** [27] --in order to represent explicitly what predicates are asserted to hold for what entities at specific times.[11] Accurate representation of the temporal structure of these situation types facilitates

[10]Webber, in work carried out in part at PRC, proposes a focusing algorithm for computing inter-sentential temporal relations which is analogous to Sidner's focusing mechanism for pronouns. Future work by Webber and Passonneau will integrate the two dimensions of temporal analysis.

[11]At present, PUNDIT's temporal component does not handle situations mentioned in modal, intensional or frequentative contexts.

efficient computation of the temporal ordering relations indicated by tense, the perfect auxiliary and temporal adverbials.

Correctly processing the temporal relation between temporal adverbials and predicated situations depends on understanding how different linguistic elements combine to refer to situations of different types. For example, both of the following sentences contain an *at*-phrase locating a situation with respect to a clock time:

1) Pressure was low at 08:00.

2) The pump seized at 08:00.

But, the relation of the clock time to the two situations differs because the first situation is *a* **state** and the second is **a transitional event.** In 1), **08:00** occurs within an interval over which the **state** of *pressure being low* holds. In 2), it coincides with a transition from a **process** of *the pump becoming seized* to a **state** of *the pump being seized.*

PUNDIT's temporal component takes into account the highly context-dependent nature of references to situations and times. Verbs, nominalizations, and other lexical items can be used predicatively to refer to situations, but their precise interpretation depends on the context in which they occur. Relevant items in the sentence context include components of the verb, such as tense, grammatical aspect and taxis,[12] as well as temporal adverbs (e.g., *before, after*)

[12] *Taxis* refers to the semantic effect of the presence or absence of the perfect auxiliary *have*. Grammatical aspect is signalled by the presence or absence of the progressive suffix *-ing* on the verb in combination with the

and aspectual verbs (e.g., *occur*, *continue*). The analysis of the temporal meaning of a specific expression is somewhat domain dependent, but the algorithm for analyzing the several sentence components which contribute to temporal reference is very general [26]. This generality has made it possible to extend the algorithm to several types of contexts, as will be described below.

The semantic decompositions represent the temporal class that a verb inherently falls in, i.e., state, process or event (see 7.2.2 above) [25]. The cooperation between the temporal component and the semantic analyzer allows PUNDIT to represent precisely what kinds of situations entities participate in and when. Its input is the semantic decomposition of the verb with its arguments filled in, as well as tense, an indication of whether the verb was in the perfect or progressive, and a list of unanalyzed constituents which may include temporal adverbials. It generates three kinds of output: an assignment of an actual time to the predication, if appropriate; a representation of the type of situation denoted by the predication (**state**, **process** or **transition event**); and finally, a set of predicates about the ordering of the time of the situation with respect to other times explicitly or implicitly mentioned in the same sentence.

11.2. General Procedure

The basic procedure for analyzing temporal reference can best be described with reference to the analysis of a simple sentence, e.g., *The sac failed.* For this

auxiliary *be*.

sentence, the input would consist of the semantic decomposition and a past tense marker:

Decomposition: become(inoperative(patient([sac1])))

Verb form: Past

The output includes a representation of a transitional event, corresponding to the *moment* of *becoming inoperative*, and a resulting state in which the sac is inoperative for some *period.* Situations are represented as predicates indicating the type of situation (e.g., **event**) with three arguments consisting of a unique identifier of the specific situation (e.g., [**fail1**]), the semantic decomposition (e.g., **becomeP(inoperativeP(patient([sac1])))**), and the time argument (e.g., **moment([fail1]).**

```
event([fail1],
        become(inoperative(patient([sac1]))),
        moment([fail1]))
state([fail2],
        inoperative(patient([sac1])),
        period([fail2]))
```

The temporal output also includes the temporal relations between situations and other known times, such as the time at which the report was filed:

precedes(moment([fail1],report_time)

starts(moment([fail1]),period([fail2])

In this case, it includes the information that the **[fail1]** event precedes the report time and that the **[fail2]** state starts immediately after the moment of the **[fail1]** event.

The relational component of a situation's temporal structure is an abstract moment of time which, following Reichenbach [32], is referred to as the **event time.** Expressing temporal relations in terms of these abstract moments greatly simplifies the computation of tense and relational adverbials. For states, the **event time** is always an arbitrary moment included within the period; for transitional events it is always the abstract moment of transition; for processes, it may be included within, or in an unspecified relation to, the period time argument.

11.3. Specific Issues

The basic algorithm used by PUNDIT's temporal component is domain independent. Built into this algorithm is the ability to consider significant contextual features during the dynamic computation of the final representation. This general ability has been tailored to the CASREPs profile in ways which will be described in this section. Further, each domain has its own characteristic vocabulary which includes specific types of temporal adverbs with and without complement phrases and clauses. Most of the temporal adverbials in the CASREPs express relations between situations (e.g., *before*, *after*, *when*).

Some of these also constrain the temporal interpretation of the adverb's complement; for example, the prepositional object of *during* necessarily has a time period associated with it. This is handled by allowing some adverbs to pass in aspectual parameters to the temporal analysis of their complements.

Some references to situations do not contain tense. PUNDIT's time component handles tenseless constituents like nominalizations and non-finite clauses by looking at the meaning and the tense of the main clause predicate. It handles tenseless sentence fragments (e.g., *erosion evident*) through default tense rules, depending on the semantics of the fragment.

The basic algorithm applies to verbs which directly denote simple situations whose participants are concrete entities (e.g., machine parts, human agents). These are referred to as first-order verbs. The algorithm also applies to verbs which are classed as second-order and third-order [26]. Second-order, or aspectual verbs, do not independently denote situations in the world but rather supply temporal information to their propositional arguments (e.g., *occur, begin, follow*). In analyzing second-order verbs, the time information contained in the verb (e.g., tense) is consulted during the analysis of the verb's arguments in order to represent the information in sentences pairs like *Sac failure occurred* and *Sac failed* in the same way. Finally, third-order verbs are those which refer to complex situations whose participants are situations, e.g., *result*, as in *contamination resulted in slow engine start.* Third-order verbs resemble first-order verbs in referring to situations, but resemble second-order

verbs by contributing temporal information pertaining to their arguments.

12. Current Application Modules

Application modules have been developed which take the general representation of the meaning of a text which PUNDIT produces and create an output tailored for the specific application. The current application modules include a module which produces a summary of the text and one which creates entries for, or queries to,, a database.

12.1. Summary

After processing a text, PUNDIT represents the message content as a set of logical forms comprising an INTEGRATED DISCOURSE REPRESENTATION (IDR). To illustrate how the system can be tailored to specific applications, a **summary component** was designed to generate a brief tabular summary of the major problems and findings mentioned in the message field of a CASREP from a selected subset of the IDR (see Figure 8).

Only three types of information contained in a complete **IDR** are required for generating a CASRREP summary. The first is the **ID List**, which contains the set of unique identifiers (IDs) to all the entities either explicitly mentioned in the text, or associated with entities mentioned in the text (e.g., **machine parts, events**). The IDs also indicate the semantic type of each entity (e.g., **id(sac,[sac1]), id(event,[fail1])**). The most significant function of the **ID List** in generating the summary is to provide a means of finding the members of a

FAILURE OF ONE OF TWO SACS. UNIT HAD LOW OUTPUT AIR PRESSURE. RESULTED IN A SLOW GAS TURBINE START. TROUBLESHOOTING REVEALED NORMAL SAC LUBE OIL PRESSURE AND TEMPERATURE. EROSION OF IMPELLOR BLADE TIP EVIDENT. CAUSE OF EROSION OF IMPELLOR BLADE UNDETERMINED. NEW SAC RECEIVED.

Status of Sac:
Part: sac State: inoperative

Damage:
Part: blade tip State: eroded

Finding:
Part: air pressure State: lowered

Finding:
Part: oil pressure State: normal

Finding:
Part: oil temperature State: normal

Finding:
Agent: ship's force State: has new sac

Figure 8.
Sample CASREP and Automatically Generated Summary

set referred to by a plural or conjoined noun phrase (e.g, *normal sac lube oil pressure and temperature*). The IDs for sets are represented in a structure of the following form:

is_group([scalars2],members(scalar,[[pressure8],[temperature1]]),numb(2))

The summary generator uses such structures to relate members of a set (e.g., **[pressure8]** and **[temperature1]**) to the unique identifier of the set itself (e.g., **[scalars2]**).

The second type of **IDR** input used in generating a summary is the **Property list**, i.e., a list of properties derived from processing adjectival and nominal modifiers of nouns. Properties may be simple unary predicates (e.g., **low([pressure7])** or more complex structures indicating the semantic decomposition of **predicating nouns**, such as

pressure([pressure3],pressureP(theme([oil5]),location([sac17])))).

The **summary component** also gets the **State list** containing the full representations of references to temporally static situations (cf. Section 9 for a definition of the distinction between **states**, **processes** and **events**). **States** can be referred to by a clause (e.g., *unit had low output air pressure*) or a nominalization (e.g., *erosion of blade tip*). Each **state** representation contains a semantic predicate decomposition with the arguments filled in (e.g., **erodedP(patient([blade1]))))**. The decompositions are logical terms whose structure resembles the terms contained in the **Property List** described in the preceding paragraph.

A summary potentially contains three information fields (cf. Figure 8): a field reporting the **Status of Sac**, and fields for **Damage** to components of the *sac* and general **Findings** pertaining to the functioning of the *sac*. All information in an **IDR** which does not fall under one of these headings is ignored. The

IDR for the sample CASREP shown in Figure 8 contains seven states. Three of these are not reported on in the summary because they represent the cognitive state of the person who filed the report; these cognitive states are derived from the three sentences with the verb *reveal* and the adjectives *evident* and *undetermined.* The four other states do pertain to the *sac* and are summarized in the four fields in Figure 9. The property list is used to help find a full description of an entity. For example, the unique identifier of the entity in the **state** representation corresponding to the phrase *erosion of impellor blade tip* is **[blade2]**:

```
state(
  [erode2]
  erodedP(patient([blade2]))
  period([erode2]))
```

But the **Property List** contains a representation of the relation between the nouns *tip* and *blade* in the prepositional phrase argument of *erosion.*

```
has_area([blade2],location([tip2]),theme([blade2]))
```

This predicate is examined by the **summary component** in order to generate the description *blade tip* for the **Part** whose **State** is *eroded.*

The **IDR** generated by PUNDIT contains all the information extracted during processing of an input text. It also serves as the representation of the current context against which each successive sentence in a text is evaluated. For any one application, the **IDR** is likely to contain too much information, but

it is complete enough to serve as input to a wide variety of application-specific post-processors.

12.2. Database Entry and Query

Another useful application for natural language text processing is automatic database entries capturing the information stored in the texts. We have created a small sample schema for a relational database to store CASREP information. This database includes relations for information about significant maintenance situations, such as failure, damage, and investigative activities; information about components involved in significant maintenance situations; and information about temporal relationships among the states and events described in the text. The database entry application takes the output from PUNDIT and reformulates it to be compatible with this database structure. For example, processing for the sentence *Sac failure occurred during engine start* results in a set of entries to the database as shown in Figure 9. (The temporal information is complex, and is omitted for clarity.) These entries indicate that a **failure** (labeled [**fail2**]) involved a sac (labeled [**sac1**]) and a situation of **normal_operation,** that is a start in this case (labeled [**start1**]), involved an engine (labeled [**engine2**]). The labels for pieces of equipment are arbitrary, and would be replaced by serial numbers or other unique identifiers in a real application. The database created from these entries can be queried using a simple database query application developed under IR&D, which provided the basis for the database entry application.

SAC failure occurred during engine start.

```
failure([fail2],fail,[sac1])
partinfo([sac1],sac)
partinfo([engine2],engine)
normal_operation([start1],start,[engine2])
```

Figure 9.
Sample CASREP sentence and Database Entries

13. The Development Environment

We have developed a set of tools designed to facilitate and make more efficient the writing, modifying and testing of arbitrary Prolog code, lexical entries, and BNF grammar rules. These tools have also proved very useful for porting PUNDIT to new domains.

13.1. Switches

It is useful to be able to tailor the operation of PUNDIT to the specific task at hand. For example, if one is working on syntax, it is possible to suppress the semantic analysis; conversely, if one is working on semantics, there is generally no need to see the results of the syntactic analysis. A SWITCHES mechanism has been implemented to enable us to modify the natural-language system environment in this way. Other switches control the use of translated code (for speed) vs. interpreted code (for debugging), parsing with or without

conjunction, and various other tracing and additional output options.

13.2. Prolog Structure Editor

The Prolog Structure Editor is a general structure editor written in Prolog and intended to facilitate the editing of Prolog terms (or sets of Prolog terms) by allowing the user to traverse the internal structure of the term being edited. We currently use the Prolog Structure Editor in order to edit grammar rules, word definitions in the lexicon, and arbitrary Prolog clauses. (In Quintus Prolog, compiled Prolog clauses cannot be edited using the Structure Editor). Regardless of the kind of term(s) being edited, editing is done using MOVEMENT commands, which allow the user to traverse the structure of a term by upward and downward movement, and EDITING commands, which enable the insertion, deletion, and replacement of arbitrary Prolog structures.

13.3. Lexical Entry Facility

Adding new vocabulary items is one of the most time-consuming aspects of porting a natural language system to a new domain. In order to reduce the overhead involved in such tasks, we have developed an interactive lexical entry procedure to guide the user in adding new vocabulary items to the lexicon. This procedure can be invoked in either one of two modes: 1) during parsing, if the lexical lookup program fails to find an entry for a word, the parser can be set to trap to the lexical entry procedure, which can then be used to create an entry

for the unknown word on the fly; and 2) the procedure can be used independently (i.e., not while parsing) to create dictionary entries for new words. The procedure elicits the relevant linguistic information from the user such as lexical category, irregular forms, and possible abbreviations; it computes dependencies among attributes, so that the user is asked for a minimum of information; it prompts for morphologically related forms (with a guess about the correct form); and it allows the user to inspect and edit all entries created. The program then automatically creates a dictionary entry from the information provided by the user. Lexical entries can also be edited after they have been created, by using the word-editing facility of the Prolog Structure Editor described above. A version of this program runs on the Symbolics Lisp Machine with a mouse-and-menu interface which eliminates most typing and frees the user from detailed formatting of dictionary entries. We expect substantial increases in the quality of the lexical entries (in terms of consistency and completeness), as well as in the efficiency with which the user can create lexical entries for new words. The lexical entry procedure is designed for users who are familiar with the PUNDIT system, but also includes help facilities for the most difficult aspects of lexical entry.

13.4. Semantic Rule Entry Procedure

The four types of semantics rules PUNDIT uses in arriving at a semantic representation are 1) lexical decompositions, 2) a set of mapping rules for

clauses and 3) a set of mapping rules for noun phrases, and finally, 4) the rules enforcing the semantic class restrictions (cf. Sections 7 and 8). Entering new sets of semantics rules during development or when porting PUNDIT to a new domain is a time-consuming and error-prone task. We have developed a program that elicits decompositions from the user, and then prompts the user for the clause mapping rules, noun phrase mapping rules, and semantic class restriction rules that are associated with the newly entered decomposition. The entry procedure ensures that each decomposition provided by the user is complete, and offers a set of options for recovery if the proposed decomposition is not complete. It also checks to see that every rule entered is consistent with already existing rules.

13.4.1. Decompositions: Define Rules

Decompositions consist of one or more **define** rules of the form **define(A,[B])**. The first argument to a **define** rule may be a lexical item (e.g., *investigate*), in which case its second argument will be a semantic predicate (e.g.,**investigateP**):

define(investigate,[investigateP(actor(A),theme(T))]).

Alternatively, the first argument to a **define** rule may be a semantic predicate (e.g., **investigateP**), in which case the second argument will either be another semantic predicate, which in turn needs a **define** rule, or it will be the empty list, indicating that the decomposition for the lexical item is complete.

Each rule entry session collects rules for a single lexical item. When the rule entry procedure is first invoked, it indicates to the user whether the lexical item already has a decomposition associated with it. If so, it allows the user to edit the existing rule, using the Prolog Structure Editor (cf. Section 12.2), by presenting the following prompt:

```
Editing a list
Element 1: define(investigate,
                  [investigateP(actor(A),theme(T))])
```

If there is no **define** rule for the lexical item, the user is prompted for one. In the sample session excerpted below, the user has given a decomposition for *monitor.*

```
No define rules exist yet for this predicate.
Input a file where all your work from this session will be noted:
  RulesFile.1

Input the decomposition of monitor:

   [monitorP(actor(A),theme(T))].
```

After each new define rule has been entered, the rule entry procedure displays the current set of define rules associated with the lexical item. A decomposition is not complete until there is a set of define rules terminating in the empty list. In this sample session, the user has entered a define rule for the lexical item *monitor,* but there is no define rule for the predicate **monitorP(actor(A),theme(T))**.

```
Your decomposition is NOT complete
     It only goes as far as: monitorP(actor(A),theme(T))

     monitor
          monitorP(actor(A),theme(T))
               *Incomplete*
```

After informing the user of the **Incomplete** status of the decomposition, the system offers a menu of the following choices:

```
What do you want to do?
1. Change the predicate that is incomplete
2. Add a define rule for the predicate that is incomplete
3. Quit
Please choose an item --
  2.
```

In this session, the user chooses menu item **2**, inputs the empty list, and is presented with the following message:

```
Here is the corrected decomposition

     monitor
          monitorP(actor(A),theme(T))
               []
```

After the user has completed the define rules for a lexical item, the procedure moves on to rules pertaining to its syntactic argument structure.

13.4.2. Syntactic Correspondences: Syntax Rules

Each thematic role in a semantic decomposition (e.g., **actor(A)** and **theme(T)** in **monitorP(actor(A),theme(T))**) has a set of surface constituent types which are potential role fillers. The correspondences between surface structure and thematic structure are encoded by mapping rules having three arguments:

```
syntax(actor(A),
       subj(S),
       X).
```

The first argument is the semantic role name, the second is the type of syntactic constituent which may provide a role filler, and the third indicates whether the rule applies to a particular semantic decomposition. The variable third argument in the sample rule given above indicates that the rule is general. The third argument of the following rule, which maps **experiencer(E)** to the prepositional object of *to*, indicates that it applies in the decomposition of the adjective *evident*, viz., **evidentP(theme(T),experiencer(B))**.

```
syntax(experiencer(B),
       pp([to,D]),
       evidentP(theme(T),experiencer(B)))
```

There are two sets of syntax mapping rules, one for clauses and one for noun phrases. The procedure first prompts the user for clause mapping rules, and then for noun phrase mapping rules. At each prompt, the user has the

option of reviewing the existing rules.

```
For the decomposition: define(monitor,[monitorP(actor(A),theme(T))]
Are there any clause mapping rules that you want to enter? (y/n/r)
  n

Are there any noun phrase mapping rules that you want to enter? (y/
  n
```

For this example, there are no new or idiosyncratic rules which apply to the verb *monitor*.

13.4.3. Selectional Constraints: Semantics Rules

The last set of rules the user is prompted for pertain to the selectional constraints on thematic role fillers. The user is told what role and what predicate to enter a selection rule for, and can request a list of the existing types of selection rules.

```
You need to add a selection rule for the role:   actor(A)
and the predicate:   monitorP
in the predicate environment:  monitorP(actor(A),theme(T))
Type 'h.' for help,
or 'n.' if you really do not want to enter a selection rule.
```

In this session, the user enters the following rule:

```
Input the rule:
  [Context,find_type(A,animate,Context)]
```

In these rules, the **Context** variable gets instantiated to the data structure carrying around the current discourse context. The context supplies information which may be needed for resolving the semantic class restriction constraint. Here the semantic restriction (**find_type**) specifies that the entity filling the **actor**(A) role of **monitorP**(**actor**(A),**theme**(T)) must be animate. A selection rule would also have to be entered for the **theme** role of **monitorP**.

Before exiting the semantic entry procedure, a list of all the rules entered during that session is displayed for the user.

14. Automated Testing Procedures for Maintaining System Stability

14.1. Purpose

PUNDIT is a very large system with functionally discrete components which must interact in highly complex ways in order to produce complete and accurate INTEGRATED DISCOURSE REPRESENTATIONS (IDRs). As the system develops, it is extremely important to ensure that these components continue to interact in the expected ways. We have developed an automated procedure to exercise the full range of phenomena handled by selected components of the system. The procedure works through several sets of test input and compares the output with normalized output files. The result of the comparison is a record of discrepancies between the newly processed output and the expected output.

14.2. Method

Several sets of test input have been specially designed for the basic grammar and the conjunction mechanism, the ISR, the semantic interpreter, the reference resolution component, and the time-analysis component. Each test set was designed by the researcher most closely associated with the development of the corresponding component.

When the testing procedure reads input from an input file, it automatically sets the relevant environment switches depending on the type of input. For example, a switch controls the printing of the full output of the temporal analysis component. While the testing procedure needs to generate this output in order to evaluate the temporal processing, it is irrelevant to the testing of, e.g., the grammar.

The testing procedure is performed once a month. It runs in background mode and directs its output to text files. Upon completion of the test, the user is alerted and can examine the text files for discrepancies and alert the appropriate research staff.

15. Future Directions

The original Statement of Work contains a four-year plan for the development of Proteus (May 1985 to May 1989). This plan has been supplemented by an updated Statement of Work for the two-year option (May 1987 to May 1989). This work is described briefly below, followed by a description of Unisys'

plans for related work in natural-language processing.

15.1. DARPA Statement of Work

The proposed research program will extend our current natural-language understanding system Proteus-I in two major areas: it will demonstrate portability, by applying Proteus to a new application domain and it will demonstrate integration of multiple knowledge sources, including syntax, semantics, pragmatics, a domain model, and both domain-specific and commonsense reasoning modules. The first six months will be devoted to completing integration of the NYU and Unisys modules and will culminate in a demonstration of a Common-Lisp system processing CASREP messages about starting air compressors, producing both a simulation of the state of equipment and a tabular summary capturing key information. During this period, we will also test a mechanism coupling the syntax and semantics modules to provide feedback to improve processing performance and accuracy. Over the course of the two-year period of the follow-on contract, we will also incorporate techniques resulting from research at SRI on common sense reasoning. We expect this to lead to increased generality and portability of our approach.

In order to demonstrate the portability of the Proteus systems, we will (also in the first six months) select a new application domain. During the remaining 18 months, we will apply and enhance our portability tools to bring up the new application; we will evaluate our tightly coupled feedback mechanism (and extend it to include pragmatics); and we will produce a demonstration

of Proteus-II running in both CommonLisp and Prolog on the new application domain. We will also continue the work at NYU on developing and testing parallel implementations of some of the modules.

15.2. Related Work

DARPA Parallel Symbolic Processing Contract (MASC)

Unisys has another DARPA contract, administered through RADC, for work on the development of a Multi-processor Architecture for Symbolic Processing (MASC). Our research in this area focuses on the development of a language JUNIPER, which combines logic and functional programming paradigms. To test the suitability of this language for AI applications, we have adapted a set of our current applications to run in JUNIPER. The two main applications are a version of the Restriction Grammar, and a semantic-network based knowledge representation system. Both of these applications are of immediate interest to the natural-language community.

One of the issues that we are currently examining is the availability of exploitable parallelism at various levels of granularity. The Restriction Grammar framework presents some interesting opportunities for exploring *application-level* parallelism, for example, in the use of OR-parallelism for the expansion of alternative definitions in the grammar. Since the MASC application work draws staff from the natural-language group, there will be significant synergy between the MASC efforts and the continued development of the

Proteus system.

Unisys-NYU Joint NSF Contract Research

Unisys and NYU currently hold a joint NSF contract through August 1987. We have also applied for a new contract, pursuing our central line of research on tools for the acquisition of domain-specific information. The current contract, *Acquisition and Use of Semantic Information for Natural Language Processing,* has addressed the issue of automating collection of domain-specific patterns of word co-occurrences ([18], in Appendix O). The work outlined in our recent proposal to NSF focuses on increasing system robustness through interactive and/or adaptive strategies based on "learning from experience". The proposal identifies two ways that a system can intelligently respond to input outside its current scope: by suggesting related input which the system *can* understand, or by asking questions which enable it to understand the novel input. The first approach provides an efficient method to make the user aware of the system's capabilities; through the second, the system gradually extends its range of competence. Both approaches rely on an ability to diagnose the failure: that is, to identify which gap or fact in the knowledge base led to the rejection of the input. Both approaches also enable the system to function despite *incomplete* domain information, a critical factor for complex domains, where "complete" domain knowledge may be unrealizable.

The research will extend Proteus to include error diagnosis and error recovery procedures integrated with knowledge acquisition tools. The modular design of Proteus will support isolation of missing information to a particular module and data source. Interactive knowledge acquisition will then supply the missing domain-specific data. This interaction, coupled with feedback from observed selection patterns, will produce a system which can "learn" from its previous experience, including both successes and failures.

Independent Research and Development in Natural Language

In addition to contract-supported research, there is research in natural language supported as part of the Unisys IR&D program. Efforts during 1987 will focus primarily on extensions to PUNDIT for use as a front-end for natural language query of databases. This will involve extending PUNDIT to handle quantifiers and various aggregation operators. It also poses interesting research issues about the relationship between the a semantic data model used in database schemata, and the domain model and semantic information needed to process natural language input.

Towards the end of 1987, we will also begin an effort in speech recognition. We plan to examine the issues involved in coupling PUNDIT to a speech recognition system, focusing on the interfaces and flow of information between specialized speech processing hardware an the rich semantic models that PUNDIT uses. It is clear that speech processing will require a more sophisticated parse

selection strategy than what we currently use. In particular, it will require the development of a weighting scheme for "correctness" of parses. This will undoubtedly have many other uses within PUNDIT, aside from the intended application in speech processing. We also expect that the speech processing work will drive the requirements for a parallel implementation of PUNDIT, to handle the enormous processing requirements imposed by speech recognition.

16. REFERENCES

[1] Harvey Abramson, Definite Clause Transiation Grammars. In *Proc. 1984 International Symposium on Logic Programming*, Atlantic City, New Jersey, Feb. 6-9, 1984, pp. 233-241.

[2] Alan Bundy, Lawrence Byrd, George Luger, Chris Mellish, and Martha Stone Palmer, Solving Mechanics Problems Using Meta-Level Inference,. In *Expert Systems in the Micro-Electronic Age*, Michie, D. (ed.), Edinburgh University Press, Edinburgh, U.K., 1979.

[3] A. Colmerauer, Metamorphosis Grammars. In *Natural Language Communication with Computers*, L. Bolc (ed.), Springer, 1978, pp. 133-189.

[4] Deborah A. Dahl, The Structure and Function of One-Anaphora in English, PhD Thesis; (also published by Indiana University Linguistics Club, 1985), University of Minnesota, 1984.

[5] Deborah A. Dahl, Focusing and Reference Resolution in PUNDIT, Presented at AAAI, Philadelphia, PA, 1986.

[6] Deborah A. Dahl, Determiners, Entities, and Contexts, Presented at Tinlap-3, Las Cruces, New Mexico, January 7-9, 1987.

[7] V. Dahl and M. McCord, Treating Co-ordination in Logic Grammars. *American Journal of Computational Linguistics* **9, No. 2,** 1983, pp. 69-91.

[8] V. Dahl, More on Gapping Grammars. In *Proc. of the International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, Japan, 1984, pp. 669-677 .

[9] John Dowding and Lynette Hirschman, Dynamic Translation for Rule Pruning in Restriction Grammar, submitted to the Second International Workshop on Natural Language Understanding and Logic Programming, Vancouver, August 17-19, 1987.

[10] C. J. Fillmore, The Case for Case. In *Universals in Linguistic Theory*, E. Bach and R. T. Harms (ed.), Holt, Rinehart, and Winston, New York, 1968.

[11] R. Grishman, N. Sager, C. Raze, and B. Bookchin, The Linguistic String Parser. *AFIPS Conference Proceedings* **43**, AFIPS Press , 1973, pp. 427-434.

[12] Jeanette K. Gundel, Zero-NP Anaphora in Russian. *Chicago Linguistic Society Parasession on Pronouns and Anaphora*, 1980.

[13] John A. Hawkins, *Definiteness and Indefiniteness*. Humanities Press, Atlantic Highlands, New Jersey, 1978.

[14] L. Hirschman and K. Puder, Restriction Grammar in Prolog. In *Proc. of the First International Logic Programming Conference*, M. Van Caneghem (ed.), Association pour la Diffusion et le Developpement de Prolog, Marseilles, 1982, pp. 85-90.

[15] L. Hirschman and K. Puder, Restriction Grammar: A Prolog Implementation. In *Logic Programming and its Applications*, D.H.D. Warren and M. VanCaneghem (ed.), 1985.

[16] L. Hirschman, Conjunction in Meta-Restriction Grammar. *J. of Logic Programming* **4**, 1986, pp. 299-328.

[17] Megumi Kameyama, Zero Anaphora: The Case of Japanese, Ph.D. thesis, Stanford University, 1985.

[18] Francois-Michel Lang and Lynette Hirschman, Improved Parsing Through Interactive Acquisition of Selectional Patterns, Forthcoming LBS Technical Memo, Logic-Based Systems Group, Unisys, Paoli, PA, May, 1987.

[19] Beth Levin and Malka Rappaport, The Formation of Adjectival Passives. *Linguistic Inquiry* **17**(4), 1986, pp. 623-661.

[20] M. Palmer, Inference Driven Semantic Analysis. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, Washington, D.C., 1983.

[21] Martha S. Palmer, A Case for Rule Driven Semantic Processing. *Proc. of the 19th ACL Conference*, June, 1981.

[22] Martha S. Palmer, Driving Semantics for a Limited Domain, Ph.D. thesis, University of Edinburgh, 1985.

[23] Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, Recovering Implicit Information, Proc. of the 24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York, August 1986.

[24] Martha S. Palmer, Deborah A. Dahl, and Rebecca J. Passonneau, Nominalizations in PUNDIT, Proc. of the 25th Annual Meeting of the Association for Computational Linguistics, Stanford University, Stanford, California, July 1987.

[25] Rebecca J. Passonneau, Designing Lexical Entries for a Limited Domain, LBS Technical Memo No. 42, Logic-Based Systems Group, Unisys, Paoli, PA, April, 1986.

[26] Rebecca J. Passonneau, A Computational Model of the Semantics of Tense and Aspect. *Journal of Computational Linguistics*, May, 1987.

[27] Rebecca J. Passonneau, Situations and Intervals, Accepted for presentation at the 25th Annual Meeting of the Assoc. for Computational Linguistics, Stanford, July, 1987.

[28] F. C. N. Pereira and D. H. D. Warren, Definite Clause Grammars for Language Analysis -- A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence* **13**, 1980, pp. 231-278.

[29] F. C. N. Pereira, Extraposition Grammars. *American Journal of Computational Linguistics* **7**, 1981, pp. 243-256.

[30] Ellen F. Prince, Toward a Taxonomy of Given-New Information. In *Radical Pragmatics*, Peter Cole (ed.), Academic Press, New York, 1981.

[31] Malka Rappaport and Beth Levin, What to do with Theta-Roles. In *Lexicon Project Working Papers*, Center for Cognitive Science, MIT, Cambridge, MA, 1986.

[32] Hans Reichenback, *Elements of Symbolic Logic*. The Free Press, New York, 1947.

[33] N. Sager and R. Grishman, The Restriction Language for Computer Grammars of Natural Language. *Communications of the ACM* **18**, 1975, pp. 390-400.

[34] N. Sager, *Natural Language Information Processing: A Computer Grammar of English and Its Applications*. Addison-Wesley, Reading, Mass., 1981.

[35] Candace Lee Sidner, Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse, MIT-AI TR-537, Cambridge, MA, 1979.

[36] Bonnie L. Webber, The Interpretation of Tense in Discourse. *To be presented at the 25th Annual Meeting of the Assoc. for Computational Linguistics*, July, 1987.

[37] Bonnie L. Webber, Event Reference, Paper presented at TINLAP-3, Las Cruces, New Mexico, January, 1987.

APPENDIX A

An Overview of the PUNDIT Text Processing System

This paper by Lynette Hirschman, Deborah Dahl, John Dowding, François Lang, Marcia Linebarger, Martha Palmer, Leslie Riley, and Rebecca Schiffman was presented at the Penn Linguistics Colloquium, Philadelphia, February, 1987.

APPENDIX B

Recovering Implicit Information

This paper by Martha S. Palmer, Deborah A. Dahl, Rebecca J. [Passonneau] Schiffman, Lynette Hirschman, Marcia Linebarger, and John Dowding, was presented at the 24th Annual Meeting of the Association for Computational Linguistics, New York, June, 1986. It describes the communication between the syntactic, semantic, and pragmatic modules that is necessary for making implicit information explicit.

APPENDIX C

Focusing and Reference Resolution in PUNDIT

This paper by Deborah Dahl was presented at AAAI-86 in Philadelphia, August, 1986. It describes the syntactic focusing algorithm used in PUNDIT, and its uses in reference resolution for pronouns, elided noun phrases, *one*-anaphora, and implicit associates.

APPENDIX D

A Dynamic Translator for Rule Pruning in Restriction Grammar

This paper by John Dowding and Lynette Hirschman has been submitted to the 2nd International Workshop on Natural Language Processing and Logic Programming, to be held in Vancouver, B.C., Canada, August 17-19, 1987.

APPENDIX E

Determiners, Entities, and Contexts

This paper by Deborah Dahl was presented at TINLAP-3, Las Cruces, New Mexico, January, 1987. It describes problems with certain indefinite noun phrases and argues that a procedure analogous to reference resolution for clauses is required to handle them.

APPENDIX F

Verb Taxonomy

This appendix by Martha Palmer gives the complete verb taxonomy for the verbs in the CASREP corpus. It then lists each verb's decomposition(s) along with the associated mapping rules and semantic class restriction rules for the semantic roles in the decompositions.

APPENDIX G

Conjunction in Meta-Restriction Grammar

This paper, by Lynette Hirschman, appeared in the **Journal of Logic Programming**, Vol. 4 (299-328). It describes the handling of conjunction in Restriction Grammar, based on the use of meta-rules. The paper describes how a number of complex conjunction problems are handled, including the scoping problems for conjoined nouns, the "comma" conjunction problem, and paired conjunctions such as *both...and.*

APPENDIX H

A Prolog Structure Editor

This paper, by Leslie Riley and John Dowding, appeared as Paoli Research Center Technical Report No. 29, January, 1986. It describes a structure editor designed to facilitate the editing of specialized constructs, such as grammar rules and lexical entries.

APPENDIX I

Designing Lexical Entries for a Limited Domain

Technical Memo No. 42, by Rebecca J. Passonneau, documents the general methods used in designing the verb decompositions and mapping rules for new domains.

APPENDIX J

A Computational Model of the Semantics of Tense and Aspect

Technical Memo No. 43, by Rebecca J. Passonneau, gives an overview of PUNDIT's temporal component. It describes the goals of the analysis, the algorithm it uses, and how the output is represented. A revised version of this paper has been submitted to the ACL to be included in a special issue on events and time.

APPENDIX K

Nominalizations in PUNDIT

This paper by Deborah Dahl, Martha Palmer, and Rebecca J. Passonneau will be presented at the 25th Annual Meeting of the Association for Computational Linguistics in Palo Alto, July 1987.

APPENDIX L

Situations and Intervals

This paper, by Rebecca J. Passonneau, will be delivered at the July 1987 meeting of the ACL. It explains the temporal structures of the three situation types--states, processes and events--and shows how they are computed.

APPENDIX M

The Interpretation of Tense in Discourse

This paper by Bonnie Webber will be presented at the 25th Annual Meeting of the Association for Computational Linguistics, Palo Alto, July, 1987.

APPENDIX N

Report on an Interaction between the Syntactic and Semantic Components

This report by Marcia Linebarger describes the design of an interaction between syntax and semantics to allow input from the semantic component to the parser to guide the parser to a semantically acceptable parse.

APPENDIX O

Improved Parsing Though Interactive Acquisition of Selectional Patterns

The report by François and Lynette Hirschman, will be issued as a Paoli Research Center Technical Report. It describes a mechanism for collecting valid sublanguage co-occurrence patterns, and their use in pruning the search focus during parsing. Through the use of selectional patterns, the average number of parses dropped from 4.7 parses/sentence to 1.5 parses/sentence.